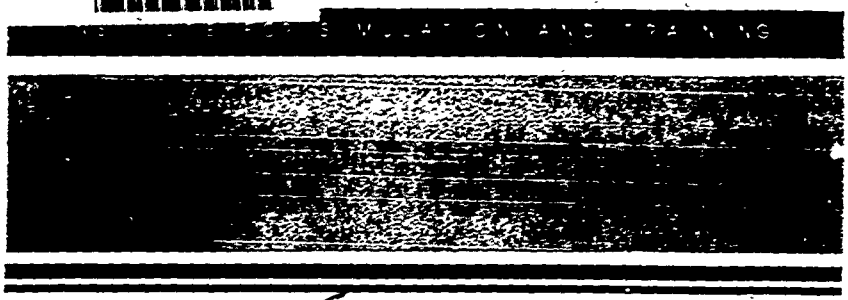


Contract Number N61339-89-C-0043

Transmission Characteristics of the

3COM ETHERLINK II NETWORK

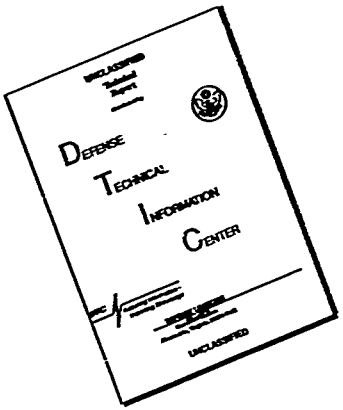
Michael Georgiopoulos
Yousuf Cheng-Hung Ma

Institute for Simulation and Training
12424 Research Parkway, Suite 300
Orlando FL 32826

University of Central Florida
Division of Sponsored Research

IST-CR-90-7

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS: NONE
2a. SECURITY CLASSIFICATION AUTHORITY: N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for Public Release; Distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): IST-CR-90-7	5. MONITORING ORGANIZATION REPORT NUMBER(S): IST-CR-90-7
6a. NAME OF PERFORMING ORGANIZATION: Institute for Simulation and Training	6b. OFFICE SYMBOL (If applicable):
6c. ADDRESS (City, State, and ZIP Code): 12424 Research Parkway, Suite 300 Orlando, FL 32826	7a. NAME OF MONITORING ORGANIZATION: Project Manager for Training Devices
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: DARPA/TTO	8b. OFFICE SYMBOL (If applicable):
8c. ADDRESS (City, State, and ZIP Code): 1400 Wilson Blvd. Arlington, VA 22209	7b. ADDRESS (City, State, and ZIP Code): 12350 Research Parkway Orlando, FL 32826
	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: N61339-89-C-0043
	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO; PROJECT NO.; TASK NO.; WORK UNIT ACCESSION NO

11. TITLE (Include Security Classification)

Transmission Characteristics of the 3COM ETHERLINK II Network

12. PERSONAL AUTHOR(S)

Georgiopoulos, Michael; Cheng-Hung Ma, Yousuf

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
TECHNICAL	FROM 4/89 TO 4/90	April 1, 1990	

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	ETHERNET, SIMNET, Protocol Enhancenents

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

In this report, the transmission characteristics of the 3Com Etherlink II adapter are examined. In particular, the transmission speed of the adapter is investigated with data passed and without data passed from host memory to the adapter buffer. Furthermore, experiments are conducted to examine the capability of replacing an old packet, submitted to the adapter buffer, with a new packet from the host memory, as well as the capability of stopping the transmission of a packet already submitted to the network adapter. The primary motivation of this investigation is to understand the behavior of the 3Com ETHERLINK II adapter in a simulation networking environment under real time constraints.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL: Mike Garnsey	22b. TELEPHONE (Include Area Code): 407-380-4816 / 22c. OFFICE SYMBOL: AMCPM-TND-EN

DD Form 1473, JUN 86 — Previous editions are obsolete. — SECURITY CLASSIFICATION OF THIS PAGE

Transmission Characteristics of the 3Com ETHERLINK II NETWORK ADAPTER

Michael Georgiopoulos
Dept. of Electrical Engineering

Yousuf Cheng-Hung Ma
Dept. of Computer Engineering

Abstract

In this report, the transmission characteristics of the 3Com Etherlink II adapter are examined. In particular, the transmission speed of the adapter is investigated with data passed and without data passed from host memory to the adapter buffer. Furthermore, experiments are conducted to examine the capability of replacing an old packet, submitted to the adapter buffer, with a new packet from the host memory, as well as the capability of stopping the transmission of a packet already submitted to the network adapter. The primary motivation of this investigation is to understand the behavior of the 3Com ETHERLINK II adapter in a simulation networking enviornment under real time constraints.

1. INTRODUCTION–MOTIVATION

The subject of this report is to understand the 3Com ETHERLINK II adapter data transmission characteristics. The 3Com Etherlink II adapter is a high performance network interface that links an IBM PC, XT, AT, PS/2 Model 25 or 30, or compatible personal computer to IEEE 802.3 Ethernet networks. The experiments conducted deal with the following items.

Experiments

1. Transmission speed of the adapter with data (packet) pass from remote memory (host memory) to local memory (adapter buffer). The program TSTRX1.ASM was used to achieve this goal.
2. Transmission speed of the adapter without data (packet) pass from remote memory to local memory. To complete this task the program TSTRX2.ASM was used.
3. The possibilty of stopping the transmission of a packet (data), already submitted to the adapter. To perform this task the program TSTRX4.ASM was utilized.
4. The possibility of replacing an old packet (data), already submitted to the adapter buffer, with a new packet (data) from the host memory. Programs TSTRX6 and TSTRX7.ASM are used to achieve this goal.

The equipment used to conduct experiments 1–4 are i) two HP Vectra RS/20C computers and ii) an HP 4972A LAN protocol analyzer. The motivation for experiments 1 and 2 is to examine the maximum transmission speed capabilities of a single network node (in this case the HP Vectra RS/20C). This information will be useful when the time comes to overload a network of HPs in order to evaluate, experimentally, the performance of the Ethernet protocol. The rationale of

experiments 3 and 4 is that, in certain real-time applications, we want to have the flexibility to replace old data with new data or stop a packet's transmission before it is completed. This capability is beneficial in voice applications, where a new voice packet (new data) replaces an unsuccessfully transmitted old packet (old data), or in SIMNET applications where a new state update message (new data) replaces an unsuccessfully transmitted old state update message (old data).

2. EXPERIMENTS 1 and 2

Experiment 1 examines the transmission speed characteristics of the adapter when data is passed from the host memory to the adapter buffer. Data is passed from the host memory on to the adapter buffer using DMA. The test environment is set up to avoid uneccessary time delays due to program instructions, data generation time and busy network channel conditions. In particular, the following constraints were imposed.

a. The data passed is a packet of fixed length.

b. The data generation time is zero.

c. The network channel is collision free (i.e., only one computer was allowed to send data).

d. The program was shortened to eliminate uneccessary instructions.

The above conditions, a-d, allow us to measure more accurately the true transmission speed of the adapter. Each time a packet is transmitted to the adapter a transmission command is issued and the program gets into a waiting mode until this packet is successfully transmitted. Three different packet lengths were tested and statistical results from the HP LAN analyzer are reported in Tables 1-3. In

the first three columns of Tables 1–3 the peak or average traffic generated by the HP computer (with data pass from host memory to adapter buffer) is shown for eight different experiment runs. The next two columns give us the total number of packets (frames) and the total number of bytes generated by the HP for the eight different runs. Finally, in the last column the average interarrival time between two consecutive packet transmissions by the HP is depicted.

	Peak or Average %	Peak or Ave. kbits/s	Peak or Ave. frms/s	Total Frames	Total Bytes	Packet elapsed Time (usec.)
1	18.40	1,826	2,536	25,335	2.280E+6	420
2	18.40	1,826	2,536	25,341	2.281E+6	380
3	18.40	1,826	2,536	25,329	2.280E+6	390
4	18.40	1,826	2,536	25,330	2.280E+6	380
5	18.40	1,826	2,536	25,348	2.281E+6	420
6	18.40	1,826	2,536	25,351	2.282E+6	380
7	18.40	1,826	2,536	25,347	2.281E+6	380
8	18.40	1,826	2,536	25,355	2.282E+6	380
Ave.	18.40	1,826	2,536	25,342	2.281E+6	392.5

Table 1 Packet length 78 bytes

	Peak or Average %	Peak or Ave. kbits/s	Peak or Ave. frms/s	Total Frames	Total Bytes	Packet elapsed Time (usec.)
1	21.31	2,114	1,716	17,140	2.640E+6	570
2	21.31	2,114	1,716	17,156	2.642E+6	570
3	21.31	2,114	1,716	17,154	2.642E+6	570
4	21.31	2,114	1,716	17,157	2.642E+6	580
5	21.31	2,114	1,716	17,156	2.642E+6	580
6	21.31	2,114	1,716	17,148	2.641E+6	570
7	21.31	2,114	1,716	17,157	2.642E+6	580
8	21.31	2,114	1,716	17,146	2.640E+6	570
Ave.	21.31	2,114	1,716	17,152	2.641E+6	573.75

Table 2 Packet length 142 bytes

	Peak or Average %	Peak or Ave. kbits/s	Peak or Ave. frms/s	Total Frames	Total Bytes	Packet elapsed Time (usec.)
1	23.77	2,358	1,045	10,443	2.945E+6	960
2	23.77	2,358	1,045	10,447	2.946E+6	960
3	23.77	2,358	1,045	10,453	2.948E+6	960
4	23.77	2,358	1,045	10,452	2.947E+6	960
5	23.77	2,358	1,045	10,444	2.945E+6	930
6	23.77	2,358	1,045	10,448	2.946E+6	960
7	23.77	2,358	1,045	10,447	2.946E+6	920
8	23.77	2,358	1,045	10,447	2.945E+6	920
Ave.	23.77	2,358	1,045	10,448	2.946E+6	946.25

Table 3 Packet length 270 bytes

Experiment 2 is similar with experiment 1 except that at at each transmission time there is no data (packet) pass from the host memory to the adapter buffer; instead the packet (data) resides in the adapter buffer and a transmission command is issued to the adapter. For experiment 2, as in experiment 1, we wait until the packet is successfully transmitted before a new transmission command is generated. The readings from the network analyzer are reported in Tables 4–6 for various packet lenght sizes. A comparison between Tables 1–3 and 4–6 shows the obvious fact that the transmission speed of the adapter with no data pass from host memory to the adapter is approximately twice its transmission speed when data pass from host memory to network adapter is involved.

	Peak or Average %	Peak or Ave. kbits/s	Peak or Ave. frms/s	Total Frames	Total Bytes	Packet elapsed Time (usec.)
1	35.86	3,558	4,942	49,426	4.448E+6	220
2	35.86	3,558	4,942	49,432	4.449E+6	200
3	35.87	3,559	4,943	49,421	4.448E+6	190
4	35.87	3,559	4,943	49,417	4.448E+6	220
5	35.87	3,559	4,943	49,418	4.448E+6	190
6	35.86	3,559	4,942	49,404	4.446E+6	200
7	35.87	3,559	4,943	49,396	4.446E+6	190
8	35.87	3,559	4,942	49,410	4.447E+6	190
Ave.	35.87	3,559	4,943	49,417	4.448E+6	200

Table 4 Packet length 78 bytes

	Peak or Average %	Peak or Ave. kbits/s	Peak or Ave. frms/s	Total Frames	Total Bytes	Packet elapsed Time (usec.)
1	48.82	4,844	3,932	39,302	6.053E+6	260
2	48.82	4,844	3,932	39,295	6.051E+6	250
3	48.81	4,843	3,931	39,290	6.051E+6	260
4	48.81	4,843	3,931	39,301	6.052E+6	260
5	48.81	4,843	3,931	39,306	6.053E+6	250
6	48.81	4,843	3,931	39,294	6.051E+6	260
7	48.81	4,843	3,931	39,287	6.050E+6	250
8	48.81	4,843	3,931	39,303	6.053E+6	250
Ave.	48.81	4,843	3,931	39,298	6.052E+6	255

Table 5 Packet length 142 bytes

	Peak or Average %	Peak or Ave. kbits/s	Peak or Ave. frms/s	Total Frames	Total Bytes	Packet elapsed Time (usec.)
1	63.98	6,348	2,814	28,122	7.930E+6	380
2	63.98	6,348	2,814	28,135	7.934E+6	350
3	63.98	6,348	2,814	28,110	7.927E+6	350
4	63.98	6,349	2,814	28,114	7.928E+6	350
5	63.98	6,348	2,814	28,135	7.934E+6	350
6	63.98	6,348	2,814	28,126	7.932E+6	350
7	63.98	6,348	2,814	28,134	7.934E+6	350
8	63.98	6,349	2,814	28,126	7.932E+6	360
Ave.	63.98	6,348	2,814	28,125	7.931E+6	355

Table 6 Packet length 270 bytes

3. EXPERIMENTS 3 and 4

This experiment has been carefully prepared to monitor every packet transmitted. Two numbers are marked on each packet. Each number is a byte on the packet data space. The first number is the 14th byte and the second number is the 16th byte. The first number corresponds to the identity of the packet that requests transmission via the adapter on to the network. A packet with identity x will be called packet x. The difference between the second number and the first number corresponds to the number of packets that have been submitted to the adapter for transmission, whose transmission process has been interrupted by a stop command.

This experiment is performed in a series of steps:

Step 1. If packet n ($n \geq 1$) is generated by the HP computer, check the bits COL, ABT, OWC or PTX of the NICSR register. (The meaning of these bits is provided in Appendix A).

Step 2. If any of the bits COL,ABT,OWC is set (=1) or the bit PTX is reset (=0), it means that the previous packet (i.e., packet $n-1$) submitted to the adapter has not been successfully transmitted yet. Then, issue a stop command and wait until it becomes effective.

Step 3. When the stop command for packet $n-1$ becomes effective submit packet n to the adapter and issue a transmit command.

A packet that is submitted to the adapter for transmission waits for two distinct time periods before its transmission is considered complete. The first time period is the interval $[t_0, t_1]$ and the second time period is the interval $[t_1, t_2]$, where the parameters t_0, t_1, t_2 have the following meaning:

1) t_0 : The time of issuing a transmit command or beginning of the waiting for the next retry. Any packet that suffers a collision returns to this point.

2) t_1 : The time at which packet transmission starts; this is the time at which the adapter starts placing the first bit of the packet on the network cable.

3) t_2 : The time at which packet transmission ends; this is the time at which the adapter places the last bit of the packet on to the network cable.

An important question to answer when we try to stop the transmission of a packet, that has already been submitted to the adapter, is when the stop command becomes effective. It was our belief, from the very beginning, that if the stop command is issued in the time period $[t_1, t_2]$ it becomes effective after t_2; in other

words we believed that we could not stop the transmission of a packet whose bits have already been placed by the adapter on to the network cable. On the contrary, we thought that if the stop command was issued in the time period $[t_0, t_1]$ we could stop the transmission of a packet already submitted to the network adapter. Due to the above observations, we conducted two experiments.

In experiment 3a, a computer (HP) sent ten packets with random interarrival time. All packets were received by the network analyzer. We observe, from experiment 3a, that the second number (16th byte) of packet arrivals 323 and 324 are the same. This means that packet 02 (i.e., packet arrival 324) came prior to the successful-transmission of packet 01 (i.e., packet arrival 323) and we tried to stop the transmission of packet 01. Since, we received packet 01 it seems that we were unable to do so. But if we observe the time difference between the arrivals of packet 02 and 01 at the network analyzer (only 400 μs) the reason might have been that packet 02 arrived in the time period $[t_1, t_2]$ of packet 01.

EXPERIMENT 3a

```
#323        Elapsed  0:00:02.92126 Len   78 Filters 0......... No error
            Destination 02-60-8C-01-02-03   Source 02-60-8C-0F-EA-88
DATA              00
  15  00-00-00-00-04-05-06-04-08-09-0A-0B-00-00-01-00-01

#324        Elapsed  0:00:02.92166 Len   78 Filters 0......... No error
            Destination 02-60-8C-01-02-03   Source 02-60-8C-0F-EA-88
DATA              00
  15  00-00-00-00-04-05-06-04-08-09-0A-0B-00-00-02-00-01

#325        Elapsed  0:00:02.92211 Len   78 Filters 0......... No error
            Destination 02-60-8C-01-02-03   Source 02-60-8C-0F-EA-88
DATA              00
  15  00-00-00-00-04-05-06-04-08-09-0A-0B-00-00-03-00-02
```

So, we performed another experiment to investigate the possibilty of stopping a packet's transmission, provided that the stop command is issued during the $[t_0, t_1]$ time period of the packet. In this experiment (experiment 3b) a computer (source 02-60-8C-0F-EA-88) produced 100 packets for transmission with random interarrival time between packets. Furthermore, another computer (source 02-CF-1F-30-27-95) sent packets continuously with random interpacket time and random packet lenghts. Let us take three packet arrivals 16,22 and 23 corresponding to consecutive packets (packets 01, 02 and 03) emanating from the same computer (source 02-60-8C-0F-EA-88). The second number (16th byte) for these packets is the same. This implies that the packet 02 came prior to the successful transmission of the packet 01 and we tried to stop the transmission of the packet 01; similarly, the packet 03 came prior to the successful transmission of packet 02 and we tried to stop the transmission of the packet 02. The difference in the arrival time between packets 03 and 02 is only 380 μs while the difference bettween the arrival times of packets 02 and 01 is approximately 0.25s. As a result, it is highly likely that the packet 03 came during the $[t_0, t_1]$ interval of packet 02. According to the experimental results we tried to stop the transmission of packet 02 and we were not successful. The stop command was issued while packet 02 was in his $[t_0, t_1]$ waiting period. In conclusion, from experiments 3a and 3b we deduce that we cannot stop the transmission of a packet that has already been submitted to the adapter for transmission.

EXPERIMENT 3b

```
#16          Elapsed  0:00:02.61190 Len   78 Filters 0......... No error
             Destination 02-60-8C-01-02-03   Source 02-60-8C-0F-EA-88
DATA                  00
  15  00-00-00-00-04-05-06-04-08-09-0A-0B-00-00-01-00-01

#17          Elapsed  0:00:02.69440 Len  102 Filters 0......... No error
             Destination FF-FF-FF-FF-FF-FF   Source 02-CF-1F-30-27-95

#18          Elapsed  0:00:02.69615 Len  102 Filters 0......... No error
             Destination FF-FF-FF-FF-FF-FF   Source 02-CF-1F-30-27-95

#19          Elapsed  0:00:02.69789 Len  102 Filters 0......... No error
             Destination FF-FF-FF-FF-FF-FF   Source 02-CF-1F-30-27-95

#20          Elapsed  0:00:02.70128 Len  102 Filters 0......... No error
             Destination FF-FF-FF-FF-FF-FF   Source 02-CF-1F-30-27-95

#21          Elapsed  0:00:02.70304 Len  102 Filters 0......... No error
             Destination FF-FF-FF-FF-FF-FF   Source 02-CF-1F-30-27-95

#22          Elapsed  0:00:02.86253 Len   78 Filters 0......... No error
             Destination 02-60-8C-01-02-03   Source 02-60-8C-0F-EA-88
DATA                  00
  15  00-00-00-00-04-05-06-04-08-09-0A-0B-00-00-02-00-01

#23          Elapsed  0:00:02.86291 Len   78 Filters 0......... No error
             Destination 02-60-8C-01-02-03   Source 02-60-8C-0F-EA-88
DATA                  00
  15  00-00-00-00-04-05-06-04-08-09-0A-0B-00-00-03-00-01
```

Experiment 4 is similar, in certain respects, with experiment 3. For example, the 14th byte of a packet (first number) indicates the packet's identity, while the difference between this number and the 16th byte (second number) indicates the number of packets whose transmission we attempted to stop. In experiment 4, we do not try to stop a packet's transmission under any circumstances. Hence, the first and the second number (14th and 16th bytes) in this experiment are always the same. In experiment 4, if the identity of the packet is an odd number we wait until the transmission of the previous packet is complete and then, we submit the packet to the adapter and we issue a transmit command; the only exception to this

rule is packet 01 for which we do not wait since it is the first packet generated by the computer. If the identity of a packet is an even number we do not wait until the completion of the previous packet's transmission, but we replace the old packet with the even numbered packet and we issue a transmit command. Our results are depicted below.

EXPERIMENT 4

```
#37          Elapsed  0:00:03.43539 Len   78 Filters 0..... No error
             Destination 02-60-8C-01-02-03   Source 02-60-8C-0F-EA-88
DATA                  00
  15  00-00-00-00-04-05-06-04-08-09-0A-0B-00-00-01-00-01

#38          Elapsed  0:00:03.43578 Len  156 Filters 0..... No error
             Destination 02-60-8C-01-02-03   Source 02-60-8C-0F-EA-88
DATA                  0C-
  15  00-00-00-00-04-05-06-04-08-09-0A-0B-00-00-04-00-04
   :        :        :        :        :
 100  07-08-09-0A-0B-0C-00-64-00-64-11-12-13-14-15-16-17

#39          Elapsed  0:00:03.43626 Len   78 Filters 0..... No error
             Destination 02-60-8C-01-02-03   Source 02-60-8C-0F-EA-88
DATA                  00
  15  00-00-00-00-04-05-06-04-08-09-0A-0B-00-00-05-00-05
```

We observe from the above results that we receive packets 01 and 04 and we miss packets 02 and 03. If the replacement action works and packet 02 came in time to replace packet 01 then, we should have received packet 02 and not packet 01. If instead packet 02 did not come in time to replace packet 01, we should have received both packets (i.e., 01 and 02). Hence, the results indicate that packet 02 came in time to replace packet 01, but the replacement action did not work. Furthermore, since the time difference between the arrival times of packet 04 and 01 is only 400

μs we are confident that packet 02 arrived in the $[t_0, t_1]$ time period of packet 01.

4. CONCLUSIONS

We conducted some experiments to compute the maximum transmission speed of the 3Com ETHERLINK II adapter with data pass and without data pass from the host (Vectra HP/20C computer) memory. We found that the adapter transmission speed is doubled if no data pass from host memory to adapter buffer is required. The complete results are shown in Tables 1-6.

We also investigated the possibilty of stopping the transmission of a packet that has already beeen submitted to the network adapter (experiment 3). Furthermore, we examined the possibility of replacing an old packet, already submitted to the adapter for transmission, with a new packet from the host memory (experiment 4). We found that, the 3Com ETHERLINK II adapter does not allow either the stopping of the transmission of a packet or the replacement of a packet already submitted to the network adapter.

APPENDIX A

Bit #	Mnemonic	Description
0	PTX	Indicates packet transmitted with no error.
2	COL	Indicates that the transmission collided at least once with another station on the network.
3	ABT	Indicates the NIC aborted transmission because of excessive collisions.
7	OWC	Indicates that a collision occurred after a slot time (51.2 us).

```
; tstrx1.asm - This program simply send packets continuously on 3C503 adapter
;              using 3L interface. Each packet transmission has packet data
;              passed from the host memory onto the adapter buffer.
;
;            ** NOTE: **  To allow this program to end cleanly
;            added savvecs and fixvecs routines to preserve vectors that
;            could possibly be changed.
;            This allows 3L interrupt hooks to be undone so 3L can be used
;            in an executable program rather than just a permanent driver.


;define 3L functions
 xtrn   InitParameters:near
extrn   InitAdapters:near
extrn   WhoAmI:near
 xtrn   ResetAdapter:near
 xtrn   RdRxFilter:near
extrn   WrRxFilter:near
xtrn    GetRxData:near
 xtrn   SetLookAhead:near
extrn   PutTxData:near

 xtrn   SetTime:near
,extrn   TimeOut:near
extrn   Ticks:word

 xtrn   Srand:near
extrn   Rand:near
xtrn    Waiting:near

public  RxProcess
public  ExitRcvInt

. so these'll be in map for debugging
public  argstr, crlf, retsav, pkthd, wbf, xmtpk, fnprmt
public  xmit1, rcvsome, dowho, savvecs, fixvecs, dmprt, prx, wtoa

lf      equ     0ah
cr      equ     0dh
minsec  equ     60d

NUMXMIT   equ    100d           ;total packets transmitted      /Ma
WAITIME   equ     16d           ;unit in usec.                  /Ma
RANDRANGE equ     11d           ;upper limit of random number   /Ma
MODUNUM   equ     10d           ;modular number with count      /Ma
XTIME10   equ     10d           ;interframe time w/pass 64 data bytes /Ma
XTIME20   equ      1d           ;     "      "     "   128  "     "   /Ma
XTIME30   equ      1d           ;     "      "     "   256  "     "   /Ma

print   macro   strloc          ;print string at strloc
        local   strloc
        push    cx
        lea     dx,strloc
        mov     ah,09h
        int     21h
        pop     cx
        endm

@kbdin  macro                   ;get kbd char in al
        mov     ah,8
```

```
        int     21h             ;wait for key
        endm

kbdchk  macro                   ;check for kbd char
        mov     ah,0bh
        int     21h             ;returns al: 0-nokey, ff-keyhit
        endm

 prx    macro   len, dat    ;print hex data in word dat, len = 1 to 4
                            ;don't put data in ax
        mov     ax,len
        push    ax
        mov     ax,dat
        push    ax
        call    prx
        add     sp,4
        endm

dmprt   macro   buf,adr,len         ;hex dump a data area
        mov     ax,len
        push    ax
        mov     ax,adr
        push    ax
        mov     ax,buf
        push    ax
        call    dmprt
        add     sp,6
        endm

CODE    GROUP   DATA, RCODE, STACK

DATA    SEGMENT WORD PUBLIC

;DOS driver init request header format
ini_hd  struc
        db      23      ;hdr len
        db      0
        db      0       ;init cmd
stat    dw      0
        db      8 dup (0)
        db      0       ;num units (not used)
cdend   dd      0       ;code end set here
argo    dw      0       ;arg offset
args    dw      0       ;arg segment
        db      0
ini_hd  ends

;---- adapter parameter setup string --------------------
;     this would come from 'device=' on real driver init
argstr  db      "bs.sys  /A:300 /D:1 /I:3",1f

;---- fake driver init request header for InitParameter input
ih      ini_hd  <,,,,,,,,offset CODE:argstr,seg CODE,>

vectsv  dd      22h dup (0)     ;save all vectors so we can cleanup

;WhoAmI adapter info structure
ad_info struc
ea      db      6 dup(0)        ;enet addr
ver1    db      0               ;major ver
```

```
[illegible]     db      ?               ;minor ver
[illegible]     db      ?               ;sub ver
[illegible]     db      ?               ;type ver
atyp    db      0               ;adapter type
astat   db      0               ;adapter status
bfrs    db      0               ;buffer fl  s
nxb     db      0               ;number of xmit buffers
sxb     dw      0               ;xmit buffer size
xmtc    dd      0               ;xmit count
xmte    dd      0               ;xmit errs
xmtto   dd      0               ;xmit timeouts
rcvc    dd      0               ;rcv count
rcvbc   dd      0               ;bcast rcv count
rcve    dd      0               ;rcv errs
rtc     dd      0               ;retry count
xfmd    db      0               ;xfer mode flags
wtmd    db      0               ;wait mode flags
extp    dw      0               ;extension pointer
ad_info ends

;program messages
crlf    db      cr,lf,'$'
TVmsg   db      "tst31 load point: $"
IPmsg   db      "InitParameters returns: $"
IAmsg   db      "InitAdapters returns: $"
WAmsg   db      "WhoAmI returns: $"
WFmsg   db      "WrRxFilter returns: $"
LAmsg   db      "SetLookAhead returns: $"
GEmsg   db      "GetRxData error return: $"
ZPmsg   db      lf,"Zero length packet",cr,lf,'$'
PAmsg   db      "Press any key to continue",cr,lf,'$'
RSmsg   db      "Starting packet receive... any key to end",cr,lf,'$'
REmsg   db      "Stopping receive",cr,lf,'$'
CLmsg   db      ":$"
HFmsg   db      " - $"
TNmsg   db      "Select function, r for recv, t for xmit: ",'$'
XMmsg   db      "Sending 1 packet",cr,lf,'$'
XRmsg   db      "PutTxData returns: $"

XMreq   db      "Transmision of packets has four options:",cr,lf
        db      "  0. Exit",cr,lf
        db      "  1. Transmit 78 byte packets without data pass.",cr,lf
        db      "  2. Transmit 142 byte packets without data pass.",cr,lf
        db      "  3. Transmit 270 byte packets without data pass.",cr,lf
;       db      "  4. Transmit the longest packet without data pass.",cr,lf
        db      cr,lf
        db      "Enter your choise: ",'$'                                       ;Ma

XMmsg1  db      "Sending 78 bytes packets for 10 seconds sampling w/D"          ;Ma
        db      cr,lf,'$'                                                       ;Ma
XMmsg2  db      "Sending 142 bytes packets for 10 seconds sampling w/D"         ;Ma
        db      cr,lf,'$'                                                       ;Ma
XMmsg3  db      "Sending 270 bytes packets for 10 seconds sampling w/D"         ;Ma
        db      cr,lf,'$'                                                       ;Ma
XMmsg4  db      "Sending long packets for one minite without data pass"         ;Ma
        db      cr,lf,'$'                                                       ;Ma

WO0msg  db      "WhoAmI DATA -",cr,lf,'$'

WO1msg  db      ".  enet addr               : $"
```

```
x.xmsg  db      "    major ver              : $"
x01msg  db      "    minor ver              : $"
x02msg  db      "    sub ver                : $"
x03msg  db      "    type ver               : $"
W06msg  db      "    adapter type           : $"
x07msg  db      "    adapter status         : $"
x08msg  db      "    buffer flags           : $"
W09msg  db      "    number of xmit buffers : $"
x10msg  db      "    xmit buffer size       : $"
x11msg  db      "    xmit count             : $"
W12msg  db      "    xmit errs              : $"
W13msg  db      "    xmit timeouts          : $"
x14msg  db      "    rcv count              : $"
x15msg  db      "    bcast rcv count        : $"
W16msg  db      "    rcv errs               : $"
x17msg  db      "    retry count            : $"
x18msg  db      "    xfer mode flags        : $"
W19msg  db      "    wait mode flags        : $"
x20msg  db      "    extension pointer      : $"

; misc parameters
retsav  dw      ?
segval  dw      ?
:off    dw      ?
errcd   db      0

pklock  db      0
pklen   dw      0
pkerr   dw      0
pkcnt   dw      0
pkcount dw      0

;avax   dw      ?

;receive buffer
pkthd   db      32 dup(0)       ;packet header portion for SetLookAhead
:pktdat  db      1500 dup(0)     ;  remainder of pkt buffer /closed by Ma

;WhoAmI buffer
wbf     ad_info <>              ;WhoAmI buffer

;****************  ready packet data ************************

'transmit 64 data byte packet
xmtpk   label   byte
desta   db      02h,60h,8ch,01h,02h,03h  ;arbitrary dest addr
;orca   db      00h,00h,00h,0fh,0fh,0fh  ;source addr - fill from who ea
plen    db      0,64                     ;packet length
pdata   db      00h,00h,00h,00h,04h,05h,06h,07h
        db      08h,09h,0ah,0bh,0ch,0dh,0eh,0fh
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
        db      20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h;34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh

.plen   dw      $-xmtpk         ;packet len

****************  ready packet data ************************
```

```
;transmit 128 data byte packet
xmtpk2  label   byte
desta2  db      02h,60h,8ch,01h,02h,03h  ;arbitrary dest addr
sorca2  db      00h,00h,00h,0fh,0fh,0fh  ;source addr - fill from who ea
plen2   db      0,128                    ;packet length
pdata2  db      00h,00h,00h,00h,04h,05h,06h,07h
        db      08h,09h,0ah,0bh,0ch,0dh,0eh,0fh
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
        db      20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h,34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh
        db      00h,01h,02h,03h,04h,05h,06h,07h
        db      08h,09h,0ah,0bh,0ch,0dh,0eh,0fh
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
        db      20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h,34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh

xplen2  dw      $-xmtpk2         ;packet len

;****************  ready packet data ************************

;transmit 256 data byte packet
xmtpk3  label   byte
desta3  db      02h,60h,8ch,01h,02h,03h  ;arbitrary dest addr
sorca3  db      00h,00h,00h,0fh,0fh,0fh  ;source addr - fill from who ea
plen3   db      0,255                    ;packet length
pdata3  db      00h,00h,00h,00h,04h,05h,06h,07h
        db      08h,09h,0ah,0bh,0ch,0dh,0eh,0fh
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
        db      20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h,34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh
        db      00h,01h,02h,03h,04h,05h,06h,07h
        db      08h,09h,0ah,0bh,0ch,0dh,0eh,0fh
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
        db      20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h,34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh
        db      00h,01h,02h,03h,04h,05h,06h,07h
        db      08h,09h,0ah,0bh,0ch,0dh,0eh,0fh
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
        db      20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h,34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh
        db      00h,01h,02h,03h,04h,05h,06h,07h
        db      08h,09h,0ah,0bh,0ch,0dh,0eh,0fh
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
```

```
        db      20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h,34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh

xplen3  dw      $-xmtpk3        ;packet len

;transmit largest packet, new data area/Ma

;xmtpk1  label   byte
;destal  db      02h,60h,8ch,01h,02h,03h  ;arbitrary dest addr
;sorcal  db      00h,00h,00h,0fh,0fh,0fh  ;source addr - fill from who ea
;plen1   dw      0,1500                   ;packet length
;pdatal  dw      187 dup (0001h,0203h,0405h,0607h,0809h,0a0bh,0c0dh,0e0fh)
;        dw      0ff11h,0ff13h

;xplen1  dw      $-xmtpk1        ;packet len

hour    db      0
min     db      0
sec     db      0
count   dw      0
funcnum db      0
sumrd   dw      0

DATA    ENDS

STACK   SEGMENT STACK
STACK   ENDS


RCODE   SEGMENT WORD PUBLIC
        assume  cs:code, ds:code

;-----------------------------------------------------------------------
;   main routine
;-----------------------------------------------------------------------
tstrx1  proc    near

        mov     ax,CODE
        mov     ds,ax
        mov     es,ax

        mov     ax,cs

        mov     segval,ax
;        mov     toff,offset CODE:tst31    ;Ma
        mov     toff,offset CODE:tstrx1    ;Ma

        @print  TVmsg               ;print prog load addr
        @prx    4,segval
        @print  CLmsg
        @prx    4,toff
        @print  crlf
        @print  PAmsg               ;wait for key
        @kbdin                      ; ... get it

        call    savvecs             ;save a bunch of vectors for later

        mov     bx,offset CODE:ih       ;fake driver init request buffer
```

```
        ; **********************
        call    InitParameters
        ; **********************
        mov     retsav,ax

        @print  IPmsg
        @prx    4,retsav
        @print  crlf
        mov     ax,retsav
        or      ax,ax
        jz      init_ok
        mov     al,1
        jmp     oout

init_ok:
        mov     di,offset CODE:RxProcess
        ; **********************
        call    InitAdapters
        ; **********************
        mov     retsav,ax

        @print  IAmsg
        @prx    4,retsav
        @print  crlf
        mov     ax,retsav
        or      ax,ax
        jz      ia_ok
        mov     errcd,2
        jmp     uninit

ia_ok:

        call    dowho           ;call WhoAmI and list result

        ; SetLookAhead is not required but added for reference
        xor     dl,dl           ;adapter 0
        mov     cx,32           ;LookAhead size
        ; **********************
        call    SetLookAhead
        ; **********************
        mov     retsav,ax

        @print  LAmsg
        @prx    4,retsav
        @print  crlf
        mov     ax,retsav
        or      ax,ax
        jz      la_ok
        mov     errcd,4
        jmp     uninit

la_ok:

        mov     pkcount,0
        xor     dl,dl           ;adapter 0
;;;     mov     ax,01h          ;set filter board address
        mov     ax,0ch          ;set filter to promis/bcast
        ; **********************
        call    WrRxFilter
        ; **********************
```

```
        mov     retsav,ax

        @print  WFmsg
        @prx    4,retsav
        @print  crlf
        mov     ax,retsav
        or      ax,ax
        jz      wf_ok
        mov     errcd,5
        jmp     uninit
wf_ok:

;---------------------------------------
        ;do xmit or rcv per user input
fnprmt:
        @print  FNmsg
        @kbdin                  ;get input selection
        push    ax
        @print  crlf
        pop     ax
        cmp     al,'r'
;       je      dorecv          ;Ma
        je      jdorecv         ;Ma
        cmp     al,'t'
        je      doxmt           ;Ma
;        jne     fnprmt         ;Ma
        jmp     fnprmt          ;Ma
jdorecv: jmp    dorecv          ;Ma

doxmt:
        @print  XMreq           ;Ma
        @kbdin                  ;Ma, get input selection
        push    ax              ;Ma
        @print  crlf            ;Ma
        pop     ax              ;Ma
        cmp     al,'1'          ;Ma
        je      doxm1           ;Ma, transmit 64 byte packets with data pass
        cmp     al,'2'          ;Ma
        je      jdoxm2          ;Ma, transmit 128 byte packets with data pass
        cmp     al,'3'          ;Ma
        je      jdoxm3          ;Ma, transmit 256 byte packets with data pass
        cmp     al,'4'          ;Ma
        je      jdoxm4          ;Ma, transmit long packets without data pass
        cmp     al,'0'          ;Ma
        je      juninit         ;Ma, end of transmision
        jne     doxmt           ;Ma
        mov     errcd,al        ;Ma
        jmp     uninit          ;Ma

jdoxm2: jmp     doxm2           ;Ma
jdoxm3: jmp     doxm3           ;Ma
jdoxm4: jmp     doxm4           ;Ma
juninit: jmp    uninit          ;Ma

; ---------------------------------------------------------------------------
: transmit 64 data byte packet continuously with data pass for 10 seconds
: sampling.
; ---------------------------------------------------------------------------
doxm1:  @print  XMmsg1          ;Ma
```

```
; ------------------------------------------------------------------------
doxm1:  @print  XMmsg1          ;Ma

        mov     count,0         ;Ma, clear count
        mov     funcnum,1       ;Ma, run function number 1
        mov     cx,NUMXMIT
repX1:
        push    cx              ;Ma
        inc     count           ;Ma
        mov     ax,count        ;Ma
        mov     byte ptr pdata[0],ah     ;Ma, mark packet number on high
        mov     byte ptr pdata[1],al     ;Ma,   .and low byte

doXmit1: call    Xmit1           ;Ma, transmit one "canned" packet

        pop     cx
        loop    repX1
        call    dowho           ;Ma, list WhoAmI result
        mov     ax,sumrd        ;Ma
        xor     dx,dx
        mov     bx,NUMXMIT
        div     bx
        mov     bx,ax
        @print  RDImsg
        @prx    4,bx
        @print  RDFmsg
        @prx    4,dx
        @print  crlf
        jmp     doxmt           ;Ma

; ------------------------------------------------------------------------
; transmit 128 data byte packet continuously without data pass for 20 seconds
; sampling.
; ------------------------------------------------------------------------
doxm2:  @print  XMmsg2          ;Ma

        mov     count,0         ;Ma, clear count
        mov     funcnum,2       ;Ma, run function number 2
        mov     cx,NUMXMIT
repX2:
        push    cx
        inc     count           ;Ma
        mov     ax,count        ;Ma
        mov     byte ptr pdata2[0],ah     ;Ma, mark packet number on high
        mov     byte ptr pdata2[1],al     ;Ma,    and low byte

doXmit2: call    Xmit1           ;Ma, transmit one "canned" packet

        pop     cx
        loop    repX2
        call    dowho           ;Ma, list WhoAmI result
        jmp     doxmt           ;Ma

; ------------------------------------------------------------------------
; transmit 256 data byte packet continuously without data pass for 10 second
; sampling
; -- ---------------------------------------------------------------------
doxm3:  @print  XMmsg3          ;Ma

        mov     count,0         ;Ma, clear count
```

```
        mov     funcnum,3       ;Ma, run function number 3
        mov     cx,NUMXMIT
repX3:
        push    cx
        inc     count           ;Ma
        mov     ax,count        ;Ma
        mov     byte ptr pdata3[0],ah     ;Ma, mark packet number on high
        mov     byte ptr pdata3[1],al     ;Ma,    and low byte

doXmit3: call    Xmit1          ;Ma, transmit one "canned" short packet

        pop     cx
        loop    repX3
        call    dowho           ;Ma, list WhoAmI result
        jmp     doxmt           ;Ma

; ---------------------------------------------------------------------------
; transmit "long canned" packet continuously without data pass for one minite
; ---------------------------------------------------------------------------
doxm4:   @print  XMmsg4         ;Ma
        jmp     doxmt           ;Ma

        call    xmit1           ;send a packet
        mov     errcd,al
        jmp     uninit

dorecv:
        call    rcvsome         ;recieve packets for till key hit
        mov     errcd,al


uninit:
        ; **********************
        call    ResetAdapter
        ; **********************
        call    fixvecs
        mov     al,errcd

oout:   mov     ah,4ch
        int     21h

;tst31   endp                   ;Ma
tstrx2  endp                    ;Ma

; ---------------------------------------------------------------
xmit1   proc    near
; ---------------------------------------------------------------
; transmit one "canned" packet
;        @print  XMmsg

        ;put our eaddr in xmit pkt
        mov     ax,word ptr wbf.ea
        mov     word ptr sorca,ax
        mov     ax,word ptr wbf.ea+2
        mov     word ptr sorca+2,ax
        mov     ax,word ptr wbf.ea+4
        mov     word ptr sorca+4,ax

        ;setup for PutTxData
;        cmp     funcnum,4d      ;Ma
```

```
;         je      set1            ;Ma
          cmp     funcnum,3d      ;Ma
          je      set3            ;Ma
          cmp     funcnum,2d      ;Ma
          je      set2            ;Ma
          cmp     count,1d        ;Ma
          jnz     notf1           ;Ma
          mov     dx,60h          ;req id and wait
          jmp     short set1      ;Ma
notf1:    mov     dx,64h          ;req id, wait and no data pass   /Ma
set1:     mov     si,offset CODE:xmtpk ;xmt pkt buffer
          mov     bx,xplen        ;set lengths
          mov     cx,bx
          jmp     setnoTx         ;Ma

set2:
          cmp     count,1d        ;Ma
          jnz     notf2           ;Ma
          mov     dx,60h          ;req id and wait
          jmp     short seto2     ;Ma
notf2:    mov     dx,61h          ;req id, wait and no data pass   /Ma

seto2:    mov     si,offset CODE:xmtpk2 ;xmt pkt buffer
          mov     bx,xplen2       ;set lengths
          mov     cx,bx
          jmp     setnoTx         ;Ma

set3:
          cmp     count,1d        ;Ma
          jnz     notf3           ;Ma
          mov     dx,60h          ;req id and wait
          jmp     short seto3     ;Ma
notf3:    mov     dx,64h          ;req id, wait and no data pass   /Ma

seto3:    mov     si,offset CODE:xmtpk3 ;xmt pkt buffer
          mov     bx,xplen3       ;set lengths
          mov     cx,bx

;set4:     mov     di,0ffffh       ;no TxProcess
setnoTx:  mov     di,0ffffh       ;no TxProcess

          ; **********************
          call    PutTxData
          ; **********************
          mov     retsav,ax

;         @print  XRmsg
;         @prx    4,retsav
;         @print  crlf
          mov     ax,retsav
          ret

xmit1     endp

; ---------------------------------------------------------------
xcvsome proc    near
; ---------------------------------------------------------------
; following code to dump received packets for a fixed time
          @print  RSmsg
chkpk:
```

```
        mov     count,0                 ;Ma, clear count
        mov     funcnum,1               ;Ma, run function number 1
        mov     cx,NUMXMIT
repX1:
        push    cx                              ;Ma

        inc     count                           ;Ma
        mov     ax,count                        ;Ma
        mov     byte ptr pdata[0],ah            ;Ma, mark packet number on high
        mov     byte ptr pdata[1],al            ;Ma,    and low byte

doXmit1: call    Xmit1          ;Ma, transmit one "canned" packet

        pop     cx
        loop    repX1
        call    dowho           ;Ma, list WhoAmI result
        jmp     doxmt           ;Ma

; ---------------------------------------------------------------------
; transmit 128 data byte packet continuously with data pass for 10 seconds
; sampling.
; ---------------------------------------------------------------------
doxm2:  @print  XMmsg2          ;Ma

        mov     count,0         ;Ma, clear count
        mov     funcnum,2       ;Ma, run function number 2
        mov     cx,NUMXMIT
repX2:
        push    cx

        inc     count                           ;Ma
        mov     ax,count                        ;Ma
        mov     byte ptr pdata2[0],ah           ;Ma, mark packet number on high
        mov     byte ptr pdata2[1],al           ;Ma     and low byte

doXmit2: call    Xmit1          ;Ma, transmit one "canned" packet

        pop     cx
        loop    repX2
        call    dowho           ;Ma, list WhoAmI result
        jmp     doxmt           ;Ma

; ---------------------------------------------------------------------
; transmit 256 data byte packet continuously with data pass for 20 second
; sampling
; ---------------------------------------------------------------------
doxm3:  @print  XMmsg3          ;Ma

        mov     count,0         ;Ma, clear count
        mov     funcnum,3       ;Ma, run function number 3
        mov     cx,NUMXMIT
repX3:
        push    cx

        inc     count                           ;Ma
        mov     ax,count                        ;Ma
        mov     byte ptr pdata3[0],ah           ;Ma, mark packet number on high
        mov     byte ptr pdata3[1],al           ;Ma,    and low byte

doXmit3: call    Xmit1          ;Ma, transmit one "canned" short packet
```

```
        ; ;     ;a
        jmp     rcpX;
        call    dowho           ;Ma, list WhoAmI result
        jmp     doxmt           ;Ma

; ---------------------------------------------------------------------
; transmit "long canned" packet continuously without data pass for one minite
; ---------------------------------------------------------------------
oxm4:   @print  XMmsg4          ;Ma
        jmp     doxmt           ;Ma

        call    xmit1           ;send a packet
        mov     errcd,al
        jmp     uninit

orecv:
        call    rcvsome         ;recieve packets for till key hit
        mov     errcd,al


uninit:
        ; **********************
        call    ResetAdapter
        ; **********************
        call    fixvecs
        mov     al,errcd

oout:   mov     ah,4ch
        int     21h

;tst31  endp                    ;Ma
 strx1  endp                    ;Ma

; ---------------------------------------------------------------
mit1    proc    near
; ---------------------------------------------------------------
; transmit one "canned" packet
:       @print  XMmsg

        ;put our eaddr in xmit pkt
        mov     ax,word ptr wbf.ea
        mov     word ptr sorca,ax
        mov     ax,word ptr wbf.ea+2
        mov     word ptr sorca+2,ax
        mov     ax,word ptr wbf.ea+4
        mov     word ptr sorca+4,ax

        ;setup for PutTxData
         cmp     funcnum,4d     ;Ma
         je      set4           ;Ma
        cmp     funcnum,3d      ;Ma
        je      set3            ;Ma
        cmp     funcnum,2d      ;Ma
        je      set2            ;Ma
        cmp     count,1d        ;Ma
        jnz     notf1           ;Ma
        mov     dx,60h          ;req id and wait
        jmp     short set1      ;Ma
iotf1:  mov     dx,64h          ;req id, wait and no data pass   /Ma
```

```
[illegible]
        [illegible] [illegible]          ;[illegible]
        [illegible] [illegible]
        jmp     setnoTx         ;Ma

set2:
        cmp     count,1d        ;Ma
        jnz     notf2           ;Ma
        mov     dx,60h          ;req id and wait
        jmp     short seto2     ;Ma
notf2:  mov     dx,64h          ;req id, wait and no data pass   /Ma

seto2:  mov     si,offset CODE:xmtpk2 ;xmt pkt buffer
        mov     bx,xplen2       ;set lengths
        mov     cx,bx
        jmp     setnoTx         ;Ma

set3:
        cmp     count,1d        ;Ma
        jnz     notf3           ;Ma
        mov     dx,60h          ;req id and wait
        jmp     short seto3     ;Ma
notf3:  mov     dx,64h          ;req id, wait and no data pass   /Ma

seto3:  mov     si,offset CODE:xmtpk3 ;xmt pkt buffer
        mov     bx,xplen3       ;set lengths
         inc     bx              ;Ma, make length 256
        mov     cx,bx

set4:    mov     di,0ffffh       ;no TxProcess
setnoTx: mov     di,0ffffh       ;no TxProcess

        ; **********************
        call    PutTxData
        ; **********************
        mov     retsav,ax

;        @print  XRmsg
;        @prx    4,retsav
;        @print  crlf
        mov     ax,retsav
        ret

xmit1   endp

; --------------------------------------------------------
rcvsome proc    near
; --------------------------------------------------------
; following code to dump received packets for a fixed time
        @print  RSmsg
chkpk:
        @kbdchk                 ;key pressed?
        or      al,al
        jz      rdbfr
        jmp     wedone
rdbfr:
        test    pklock,0ffh     ;got a pkt?
        jnz     lstpkt
        jmp     chkpk
lstpkt:
```

```
        test    pkerr,0ffffh    ;rx, crc e
        z       dmpk
        @print  CRmsg
        @prx    1,pkerr
        @print  crlf
        mov     pklock,0
        inc     pkcnt
        jmp     chkpk
dmpk:
        cmp     pklen,0
        jnz     pkok
        @print  ZPmsg
        mov     pklock,0
        inc     pkcnt
        jmp     chkpk
pkok:
        cmp     pklen,256
        jle     dmok1
        mov     pklen,256       ;limit dump to 1st 256 bytes
dmok1:
        @dmprt  <offset CODE:pkthd>,0,pklen
        mov     pklock,0
        inc     pkcnt
        jmp     chkpk

wedone:
        @print  REmsg
        mov     ax,0        ;a return code
        ret

rcvsome endp


; ---------------------------------------------------------------
;       RxProcess
; ---------------------------------------------------------------
RxProcess proc      near

        push    bx
        push    cx

        test    cs:pklock,0ffh
        jz      getp
dontget:
        inc     pkcount
        mov     cx,0        ;zero length (just discard)
        jmp     goget
getp:
        ; At this point we could check es:di packet header data
        ; to make some decision on packet disposition

        ; lock our buffer and get packet data into it
        mov     cs:pklock,0ffh ;lock buff
        mov     cs:pkerr,0
goget:
        mov     ax,CODE
        mov     es,ax
        mov     di,offset CODE:pkthd     ;buffer
        or      dl,40h      ;release buffer
        ; **********************
```

```
        call    GetRxData
        ; **********************
        jcxz    nolen
        mov     cs:pkerr,ax
        mov     cs:pklen,cx

 olen:
        pop     cx
        pop     bx
        ret
RxProcess endp

  ---------------------------------------------------------------
;       ExitRcvInt
; ---------------------------------------------------------------
xitRcvInt  proc    near

        iret

xitRcvInt  endp

; ---------------------------------------------------------------
  --- get and print WhoAmI statistics ---
  ---------------------------------------------------------------
dowho   proc    near

        push    es
        xor     dl,dl           ;adapter 0
        ; **********************
        call    WhoAmI
        ; **********************
        mov     retsav,ax

        @print  WAmsg
        @prx    4,retsav
        @print  crlf
        mov     ax,retsav
        or      ax,ax
        jz      wa_ok
        mov     errcd,3
        jmp     uninit

a_ok:
        mov     si,di
        mov     di,offset CODE:wbf
        push    ds

        push    ds
        push    es
        pop     ds
        pop     es
        mov     cx,24
        cld
   rep movsw                    ;copy who buffer

        pop     ds
        pop     es

        call    whodat          ;print the WhoAmI data
```

```
        @print  CRLF
        mov     ah,8
        int     21h             ;wait for key

        ret
whowho  endp


;-----  print WhoAmI data  ------------------------
whodat  PROC    near
        @print  W00msg

;;;;     @dmprt  <offset CODE:wbf>,0,48

        @print  W01msg
        mov     cx,6
        mov     bx,0
prtea:
        push    bx
        @prx    2,<word ptr [bx+offset CODE:wbf.ea-1]>
        pop     bx
        inc     bx
        loop    prtea
        @print  crlf

        @print  W02msg
        @prx    2,<word ptr wbf.ver1-1>
        @print  crlf

        @print  W03msg
        @prx    2,<word ptr wbf.ver2-1>
        @print  crlf

        @print  W04msg
        @prx    2,<word ptr wbf.ver3-1>
        @print  crlf

        @print  W05msg
        @prx    2,<word ptr wbf.ver4-1>
        @print  crlf

        @print  W06msg
        @prx    2,<word ptr wbf.atyp-1>
        @print  crlf

        @print  W07msg
        @prx    2,<word ptr wbf.astat-1>
        @print  crlf

        @print  W08msg
        @prx    2,<word ptr wbf.bfrs-1>
        @print  crlf

        @print  W09msg
        @prx    2,<word ptr wbf.nxb-1>
        @print  crlf

        @print  W10msg
        @prx    4,<word ptr wbf.sxb>
```

```
        @print  crlf

        @print  W11msg
        @prx    4,<word ptr wbf.xmtc+2>
        @prx    4,<word ptr wbf.xmtc>
        @print  crlf

        @print  W12msg
        @prx    4,<word ptr wbf.xmte+2>
        @prx    4,<word ptr wbf.xmte>
        @print  crlf

        @print  W13msg
        @prx    4,<word ptr wbf.xmtto+2>
        @prx    4,<word ptr wbf.xmtto>
        @print  crlf

        @print  W14msg
        @prx    4,<word ptr wbf.rcvc+2>
        @prx    4,<word ptr wbf.rcvc>
        @print  crlf

        @print  W15msg
        @prx    4,<word ptr wbf.rcvbc+2>
        @prx    4,<word ptr wbf.rcvbc>
        @print  crlf

        @print  W16msg
        @prx    4,<word ptr wbf.rcve+2>
        @prx    4,<word ptr wbf.rcve>
        @print  crlf

        @print  W17msg
        @prx    4,<word ptr wbf.rtc+2>
        @prx    4,<word ptr wbf.rtc>
        @print  crlf

        @print  W18msg
        @prx    2,<word ptr wbf.xfmd-1>
        @print  crlf

        @print  W19msg
        @prx    2,<word ptr wbf.wtmd-1>
        @print  crlf

        @print  W20msg
        @prx    4,<word ptr wbf.extp>
        @print  crlf

        ret
whodat  endp

;-------------------------------------------------------------------------
savvecs proc    near
        push    ds
        push    es
        push    si
        push    di
        push    cx
```

```
        mov     ax,ds
        mov     es,ax
        xor     ax,ax
        mov     ds,ax
        mov     cx,22h*2        ;vectors 0 - 21h, 2 wds per
        mov     di,offset CODE:vectsv
        xor     si,si
        cld
        cli
   rep  movsw                   ;save 'em all
        sti

        pop     cx
        pop     di
        pop     si
        pop     es
        pop     ds
        ret
savvecs endp

;-------------------------------------------------------------------------
fixvecs proc    near
        push    es
        push    si
        push    di
        push    cx

        xor     ax,ax
        mov     es,ax
        mov     cx,22h*2        ;vectors 0 - 21h, 2 wds per
        mov     si,offset CODE:vectsv
        xor     di,di
        cld
        cli
   rep  movsw                   ;restore 'em all
        sti

        pop     cx
        pop     di
        pop     si
        pop     es
        ret
fixvecs endp

;-------------------------------------------------------------------------
;   dmprt - produces dump listing, calling parameters are pushed on stack
;           (converted from a C routine)
;   INPUTS:
;       [bp+4]  = data address
;       [bp+6]  = starting address for line headers
;       [bp+8]  = length of data to print
;   OUTPUT:
;       Dump listing to stdout device
;-------------------------------------------------------------------------
dmprt   proc    near

        push    bp
        mov     bp,sp
        mov     bx,bp
        sub     bx,0ch          ;local vars
```

```
        mov     sp,bx
        push    si
        mov     ax,[bp+8]       ;len

d005c:  sub     dx,dx
        mov     cx,10h

d0061:  div     cx
        mov     [bp-4],ax       ;lines

d0063:  mov     [bp-6],dx       ;rem

d0066:  mov     word ptr [bp-8],0       ;i


d006b:  mov     word ptr [bp-0ah],0     ;line


d0070:  jmp     d0158


d0073:
        push    dx
        mov     dl,cr           ;000d
        mov     ah,2
        int     21h
        mov     dl,1f           ;000A
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

        mov     ax,4
        push    ax
        mov     ax,[bp+6]       ;adr
        add     ax,[bp-8]       ;i
        push    ax
        call    prx
        add     sp,4            ;0004
        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

        mov     word ptr [bp-0ch],0     ;j


d00c5:  test    byte ptr [bp-0ch],3     ;j
        jnz     d00d5
        push    dx
        mov     dl,' '
```

```
        mov     ah,2
        int     21h
        pop     dx

d00d5:  mov     ax,2            ;0002
        push    ax
        mov     bx,[bp-8]       ;i
        mov     si,[bp+4]       ;buf
        mov     ah,[bx+si]      ;buf[i]
        push    ax
        call    prx
        add     sp,4            ;0004
        inc     word ptr [bp-8] ;i
        inc     word ptr [bp-0ch]    ;j

d00f0:  cmp     word ptr [bp-0ch],10h    ;j
        jb      d00c5

        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

        sub     word ptr [bp-8],10h     ;i,0010
        mov     word ptr [bp-0ch],0      ;j


        ;do ascii
d0113:  mov     bx,[bp-8]       ;i
        mov     si,[bp+4]       ;buf
        push    dx
        mov     dl,[bx+si]      ;buf[i]
        cmp     dl,' '
        jb      d013f
        cmp     dl,7fh
        jb      d0142

d013f:  mov     dl,'.'          ;002e

d0142:
        mov     ah,2
        int     21h
        pop     dx

        inc     word ptr [bp-8] ;i
        inc     word ptr [bp-0ch]    ;j
        cmp     word ptr [bp-0ch],10h    ;0010
        jb      d0113
        inc     word ptr [bp-0ah]    ;line

d0158:  mov     ax,[bp-4]       ;lines
        cmp     [bp-0ah],ax      ;line
        jnb     d0163
        jmp     d0073
```

```
d01c3:  cmp     word ptr [bp-6],0       ;cm
        jnz     d016c
        jmp     d0272


d016c:
        push    dx
        mov     dl,cr                   ;000d
        mov     ah,2
        int     21h
        mov     dl,1f                   ;000a
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

        mov     ax,4                    ;0008
        push    ax
        mov     ax,[bp+6]               ;adr
        add     ax,[bp-8]               ;i
        push    ax
        call    prx
        add     sp,4                    ;0004
        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

        mov     word ptr [bp-0ch],0     ;j

        jmp     short   d01c3


d0198:  test    byte ptr [bp-0ch],3     ;j
        jnz     d01a8
        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

d01a8:  mov     ax,2                    ;0002
        push    ax
        mov     bx,[bp-8]               ;i
        mov     si,[bp+4]               ;buf
        mov     ah,[bx+si]              ;buf[i]
        push    ax
        call    prx
        add     sp,4                    ;0004
        inc     word ptr [bp-8] ;i
        inc     word ptr [bp-0ch]       ;j
```

```
d01ef:  mov     ax,[bp-6]       ;rem
        cmp     [bp-0ch],ax      ;j
        jb      d0198
        jmp     short   d01f4


d01cd:  test    byte ptr [bp-0ch],3  ;j
        jnz     d01dd
        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

d01dd:
        push    dx
        mov     dl,'.'
        mov     ah,2
        int     21h
        mov     dl,'.'
        mov     ah,2
        int     21h
        pop     dx

        inc     word ptr [bp-0ch]    ;j

d01f4:  cmp     word ptr [bp-0ch],10h    ;0010
        jb      d01cd
        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

        mov     ax,[bp-6]       ;rem
        sub     [bp-8],ax       ;i
        mov     word ptr [bp-0ch],0  ;j


        ;do ascii
d0219:  mov     ax,[bp-6]       ;rem
        cmp     [bp-0ch],ax      ;j
        jnb     d026c
        mov     bx,[bp-8]       ;i
        mov     si,[bp+4]       ;buf
        push    dx
        mov     dl,[bx+si]      ;buf[i]
        cmp     dl,' '
        jb      d024d
        cmp     dl,7fh
        jb      d0250

d024d:  mov     dl,'.'          ;002e

d0250:
        mov     ah,2
```

```
        int     21h
        pop     dx

        inc     word ptr [bp-8] ;i
        inc     word ptr [bp-0ch]    ;j
        jmp     short   d0219


d025f:
        push    dx
        mov     dl,'.'
        mov     ah,2
        int     21h
        pop     dx

        inc     word ptr [bp-0ch]    ;j

d026c:  cmp     word ptr [bp-0ch],10h    ;0010
        jb      d025f

d0272:
        push    dx
        mov     dl,cr           ;000d
        mov     ah,2
        int     21h
        mov     dl,lf           ;000a
        mov     ah,2
        int     21h
        pop     dx

        pop     si
        mov     sp,bp
        pop     bp
        ret
dmprt   endp

;-------------------------------------------------------------------------
;   prx - routine to print a hex value from binary data up to word length
;   INPUTS:
;       [bp+4]  = binary data to convert
;       [bp+6]  = number of bytes to print (1 to 4)
;
;-------------------------------------------------------------------------
prx     proc    near

        push    bp
        mov     bp,sp
        mov     bx,bp
        sub     bx,4            ;local space
        mov     sp,bx

        push    si
        push    dx
        push    cx
        push    ds
        mov     ax,ss           ;make temp buf accessable
        mov     ds,ax
        lea     bx,[bp-4]       ;temp buffer address
        mov     dx,[bp+4]       ;data to cvrt
        call    wtoa
```

```
        mov     cx,[bp+6]       ;char count to print
        xor     si,si
prx1:
        mov     dl,[bp+si-4]    ;get a byte
        mov     ah,2
        int     21h             ;print it
        inc     si
        loop    prx1

        pop     ds
        pop     cx
        pop     dx
        pop     si
        mov     sp,bp
        pop     bp
        ret
prx     endp


;-----------------------------------------------------------------
;       CONVERT WORD TO ASCII HEX
;    Calling sequence:
;       mov     dx,word         ;word to convert
;       mov     bx,offset out   ;where to put output
;       call    wtoa
;
;       ds:bx  needs 4 bytes for result
;-----------------------------------------------------------------

wtoa    proc    near
        push    ax
        push    bx
        push    cx
        push    dx
        push    si
        mov     si,4            ;digits per word
wtoa01:
        mov     al,dl           ;get a digit
        mov     cl,4
        shr     dx,cl           ;strip the digit
        and     al,0fh          ;keep low nibble
        add     al,090h
        daa
        adc     al,040h
        daa
        dec     si              ;count the digit
        mov     [bx+si],al      ;store the digit
        jnz     wtoa01
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        ret
wtoa    endp

RCODE   ENDS
        END tstrx1              ;Ma
```

```
; [illegible]
;           on [illegible]
;           between two [illegible]
;           ** NOTE: **  To allow this program to end cleanly
;           added savvecs and fixvecs routines to preserve vectors that
;           could possibly be changed.
;           This allows 3L interrupt hooks to be undone so 3L can be used
;           in an executable program rather than just a permanent driver.
;

;define 3L functions
extrn   InitParameters:near
extrn   InitAdapters:near
extrn   WhoAmI:near
extrn   ResetAdapter:near
extrn   RdRxFilter:near
extrn   WrRxFilter:near
extrn   GetRxData:near
extrn   SetLookAhead:near
extrn   PutTxData:near

extrn   SetTime:near
;extrn   TimeOut:near
extrn   Ticks:word

extrn   Srand:near
extrn   Rand:near
extrn   Waiting:near

public  RxProcess
public  ExitRcvInt

; so these'll be in map for debugging
public  argstr, crlf, retsav, pkthd, wbf, xmtpk, fnprmt
public  xmitl, rcvsome, dowho, savvecs, fixvecs, dmprt, prx, wtoa,sumrd

lf          equ     0ah
cr          equ     0dh
minsec      equ     60d

NUMXMIT     equ   1000d           ;total packets transmitted       /Ma
RANDRANGE   equ     11d           ;upper limit of random number    /Ma
MODUNUM     equ     10d           ;modular number with count       /Ma
FTIME10     equ    392d           ;interframe time w/pass 64 data bytes /Ma
FTIME20     equ    574d           ;    "     "    "   128  "    "  /Ma
FTIME30     equ    946d           ;    "     "    "   256  "    "  /Ma
FTIME11     equ    200d           ;    "     "  wout/pass 64 data b. /Ma
FTIME21     equ    255d           ;    "     "     "     128  "  "  /Ma
FTIME31     equ    355d           ;    "     "     "     256  "  "  /Ma

@print  macro   strloc            ;print string at strloc
        local   strloc
        push    cx
        lea     dx,strloc
        mov     ah,09h
        int     21h
        pop     cx
        endm

@kbdin  macro                     ;get kbd char in al
```

```
        endm

@kbdchk macro                   ;check for kbd char
        mov     ah,0bh
        int     21h             ;returns al: 0-nokey, ff-keyhit
        endm

@prx    macro   len, dat    ;print hex data in word dat, len = 1 to 4
                            ;don't put data in ax
        mov     ax,len
        push    ax
        mov     ax,dat
        push    ax
        call    prx
        add     sp,4
        endm

@dmprt  macro   buf,adr,len         ;hex dump a data area
        mov     ax,len
        push    ax
        mov     ax,adr
        push    ax
        mov     ax,buf
        push    ax
        call    dmprt
        add     sp,6
        endm

CODE    GROUP   DATA, RCODE, STACK

DATA    SEGMENT WORD PUBLIC

;DOS driver init request header format
ini_hd  struc
        db      23      ;hdr len
        db      0
        db      0       ;init cmd
stat    dw      0
        db      8 dup (0)
        db      0       ;num units (not used)
cdend   dd      0       ;code end set here
argo    dw      0       ;arg offset
args    dw      0       ;arg segment
        db      0
ini_hd  ends

;---- adapter parameter setup string --------------------
;     this would come from 'device=' on real driver init
argstr  db      "bs.sys  /A:300 /D:1 /I:3",1f

;---- fake driver init request header for InitParameter input
ih      ini_hd  <,,,,,,,offset CODE:argstr,seg CODE,>

vectsv  dd      22h dup (0)     ;save all vectors so we can cleanup

;WhoAmI adapter info structure
ad_info struc
ea      db      6 dup(0)        ;enet addr
```

```
ver1    db      0               ;major ver
ver2    db      0               ;minor ver
ver3    db      0               ;sub ver
ver4    db      0               ;type ver
atyp    db      0               ;adapter type
astat   db      0               ;adapter status
bfrs    db      0               ;buffer flags
nxb     db      0               ;number of xmit buffers
sxb     dw      0               ;xmit buffer size
xmtc    dd      0               ;xmit count
xmte    dd      0               ;xmit errs
xmtto   dd      0               ;xmit timeouts
rcvc    dd      0               ;rcv count
rcvbc   dd      0               ;bcast rcv count
rcve    dd      0               ;rcv errs
rtc     dd      0               ;retry count
xfmd    db      0               ;xfer mode flags
wtmd    db      0               ;wait mode flags
sxtp    dw      0               ;extension pointer
ad_info ends

;program messages
crlf    db      cr,lf,'$'
TVmsg   db      "tst31 load point: $"
IPmsg   db      "InitParameters returns: $"
IAmsg   db      "InitAdapters returns: $"
WAmsg   db      "WhoAmI returns: $"
WFmsg   db      "WrRxFilter returns: $"
LAmsg   db      "SetLookAhead returns: $"
GEmsg   db      "GetRxData error return: $"
ZPmsg   db      lf,"Zero length packet",cr,lf,'$'
PAmsg   db      "Press any key to continue",cr,lf,'$'
RSmsg   db      "Starting packet receive... any key to end",cr,lf,'$'
REmsg   db      "Stopping receive",cr,lf,'$'
CLmsg   db      ":$"
TFmsg   db      " - $"
TNmsg   db      "Select function, r for recv, t for xmit: ",'$'
XMmsg   db      "Sending 1 packet",cr;lf,'$'
XRmsg   db      "PutTxData returns: $"

RDImsg  db      "Average integer: $"
RDFmsg  db      "   Average fraction: $"

XMreq   db      "Transmision of packets has four options:",cr,lf
        db      "  0. Exit",cr,lf
        db      "  1. Transmit 78 byte packets with random time data pass.",cr,l
        db      "  2. Transmit 142 byte packets with random time data pass.",cr,
        db      "  3. Transmit 270 byte packets with random time data pass.",cr,
        db      cr,lf
        db      "Enter your choise: ",'$'                               ;Ma

XMmsg1  db      "Sending 78 bytes packets for 10 seconds sampling w/D"  ;Ma
        db      cr,lf,'$'                                               ;Ma
XMmsg2  db      "Sending 142 bytes packets for 10 seconds sampling w/D" ;Ma
        db      cr,lf,'$'                                               ;Ma
XMmsg3  db      "Sending 270 bytes packets for 10 seconds sampling w/D" ;Ma
        db      cr,lf,'$'                                               ;Ma
XMmsg4  db      "Sending long packets for one minite without data pass"  ;Ma
        db      cr,lf,'$'                                               ;Ma
```

```
[illegible]  db      "WhoAmI DATA -",cr,lf,'$'

W01msg  db      "    enet addr              : $"
W02msg  db      "    major ver              : $"
W03msg  db      "    minor ver              : $"
W04msg  db      "    sub ver                : $"
W05msg  db      "    type ver               : $"
W06msg  db      "    adapter type           : $"
W07msg  db      "    adapter status         : $"
W08msg  db      "    buffer flags           : $"
W09msg  db      "    number of xmit buffers : $"
W10msg  db      "    xmit buffer size       : $"
W11msg  db      "    xmit count             : $"
W12msg  db      "    xmit errs              : $"
W13msg  db      "    xmit timeouts          : $"
W14msg  db      "    rcv count              : $"
W15msg  db      "    bcast rcv count        : $"
W16msg  db      "    rcv errs               : $"
W17msg  db      "    retry count            : $"
W18msg  db      "    xfer mode flags        : $"
W19msg  db      "    wait mode flags        : $"
W20msg  db      "    extension pointer      : $"

; misc parameters
retsav  dw      ?
segval  dw      ?
toff    dw      ?
errcd   db      0

pklock  db      0
pklen   dw      0
pkerr   dw      0
pkcnt.  dw      0
pkcount dw      0

savax   dw      ?

;receive buffer
pkthd   db      32 dup(0)       ;packet header portion for SetLookAhead
:pktdat  db      1500 dup(0)     ;  remainder of pkt buffer /closed by Ma

;WhoAmI buffer
wbf     ad_info <>              ;WhoAmI buffer

;***************  ready packet data ***********************

:transmit 64 data byte packet
xmtpk   label   byte
desta   db      02h,60h,8ch,01h,02h,03h  ;arbitrary dest addr
sorca   db      00h,00h,00h,0fh,0fh,0fh  ;source addr - fill from who ea
plen    db      0,64                     ;packet length
pdata   db      00h,00h,00h,00h,04h,05h,06h,07h
        db      08h,09h,0ah,0bh,0ch,0dh,0eh,0fh
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
        db      20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h,34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh
```

```
xplen   dw      $-xmtpk         ;packet len

;*****************  ready packet data ************************

;transmit 128 data byte packet
xmtpk2  label   byte
desta2  db      02h,60h,8ch,01h,02h,03h  ;arbitrary dest addr
sorca2  db      00h,00h,00h,0fh,0fh,0ff  ;source addr - fill from who ea
plen2   db      0,128                    ;packet length
pdata2  db      00h,00h,00h,00h,04h,05h,06h,07h
        db      08h,09h,0ah,0bh,0ch,0dh,0eh,0fh
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
        db      20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h,34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh
        db      00h,01h,02h,03h,04h,05h,06h,07h
        db      08h,09h,0ah,0bh,0ch,0dh,0eh,0fh
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
        db      20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h,34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh

xplen2  dw      $-xmtpk2        ;packet len

;*****************  ready packet data ************************

;transmit 256 data byte packet
xmtpk3  label   byte
desta3  db      02h,60h,8ch,01h,02h,03h  ;arbitrary dest addr
sorca3  db      00h,00h,00h,0fh,0fh,0fh  ;source addr - fill from who ea
plen3   db      0,255                    ;packet length
pdata3  db      00h,00h,00h,00h,04h,05h,06h,07h
        db      08h,09h,0ah,0bh,0ch,0dh,0eh,0fh
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
        db      20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h,34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh
        db      00h,01h,02h,03h,04h,05h,06h,07h
        db      08h,09h,0ah,0bh,0ch,0dh,0eh,0fh
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
        db      20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h,34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh
        db      00h,01h,02h,03h,04h,05h,06h,07h
```

```
        db      08h,09h,0ah,0bh,0ch,0dh,0eh,0fh
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
        db      20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h,34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh

;plen3  dw      $-xmtpk3        ;packet len

;transmit largest packet, new data area/Ma

 xmtpkl  label   byte
;destal  db      02h,60h,8ch,01h,02h,03h  ;arbitrary dest addr
;sorcal  db      00h,00h,00h,0fh,0fh,0fh  ;source addr - fill from who ea
 plenl   dw      0,1500                   ;packet length
;pdatal  dw      187 dup (0001h,0203h,0405h,0607h,0809h,0a0bh,0c0dh,0e0fh)
;        dw      0ff11h,0ff13h

;xplenl  dw      $-xmtpkl        ;packet len

hour    db      0
min     db      0
sec     db      0
count   dw      0
;uncnum db      0
;umrd   dw      0

DATA    ENDS

STACK   SEGMENT STACK
STACK   ENDS


RCODE   SEGMENT WORD PUBLIC
        assume  cs:code, ds:code

;-----------------------------------------------------------------
;   main routine
;-----------------------------------------------------------------
cstrx2  proc    near

        mov     ax,CODE
        mov     ds,ax
        mov     es,ax

        mov     ax,cs

        mov     segval,ax
;        mov     toff,offset CODE:tst31     ;Ma
        mov     toff,offset CODE:tstrx2    ;Ma

        @print  TVmsg                   ;print prog load addr
        @prx    4,segval
        @print  CLmsg
        @prx    4,toff
        @print  crlf
        @print  PAmsg
        @kbdin                          ;wait for key
                                        ; ... get it
```

```
        ; **********************
        call    WrRxFilter
        ; **********************
        mov     retsav,ax

        @print  WFmsg
        @prx    4,retsav
        @print  crlf
        mov     ax,retsav
        or      ax,ax
        jz      wf_ok
        mov     errcd,5
        jmp     uninit
wf_ok:

;-------------------------------------
        ;do xmit or rcv per user input
fnprmt:
        @print  FNmsg
        @kbdin                  ;get input selection
        push    ax
        @print  crlf
        pop     ax
        cmp     al,'r'
;       je      dorecv          ;Ma
        je      jdorecv         ;Ma
        cmp     al,'t'
        je      doxmt           ;Ma
;        jne     fnprmt         ;Ma
        jmp     fnprmt          ;Ma
jdorecv: jmp    dorecv          ;Ma

doxmt:
        @print  XMreq           ;Ma
        @kbdin                  ;Ma, get input selection
        push    ax              ;Ma
        @print  crlf            ;Ma
        pop     ax              ;Ma
        cmp     al,'1'          ;Ma
        je      doxm1           ;Ma, transmit 64 byte packets with data pass
        cmp     al,'2'          ;Ma
        je      jdoxm2          ;Ma, transmit 128 byte packets with data pass
        cmp     al,'3'          ;Ma
        je      jdoxm3          ;Ma, transmit 256 byte packets with data pass
        cmp     al,'4'          ;Ma
        je      jdoxm4          ;Ma, transmit long packets without data pass
        cmp     al,'0'          ;Ma
        je      juninit         ;Ma, end of transmision
        jne     doxmt           ;Ma
        mov     errcd,al        ;Ma
        jmp     uninit          ;Ma

jdoxm2: jmp     doxm2           ;Ma
jdoxm3: jmp     doxm3           ;Ma
jdoxm4: jmp     doxm4           ;Ma
juninit: jmp    uninit          ;Ma

; ---------------------------------------------------------------------------
; transmit 64 data byte packet continuously without data pass for 10 seconds
; sampling.
```

```
        call    savvecs                 ; save a bunch of vectors for later

        mov     bx,offset CODE:ih       ;fake driver init request buffer
        ; **********************
        call    InitParameters
        ; **********************
        mov     retsav,ax

        @print  IPmsg
        @prx    4,retsav
        @print  crlf
        mov     ax,retsav
        or      ax,ax
        jz      init_ok
        mov     al,1
        jmp     oout

init_ok:
        mov     di,offset CODE:RxProcess
        ; **********************
        call    InitAdapters
        ; **********************
        mov     retsav,ax

        @print  IAmsg
        @prx    4,retsav
        @print  crlf
        mov     ax,retsav
        or      ax,ax
        jz      ia_ok
        mov     errcd,2
        jmp     uninit

ia_ok:

        call    dowho           ;call WhoAmI and list result

        ; SetLookAhead is not required but added for reference
        xor     dl,dl           ;adapter 0
        mov     cx,32           ;LookAhead size
        ; **********************
        call    SetLookAhead
        ; **********************
        mov     retsav,ax

        @print  LAmsg
        @prx    4,retsav
        @print  crlf
        mov     ax,retsav
        or      ax,ax
        jz      la_ok
        mov     errcd,4
        jmp     uninit

la_ok:

        mov     pkcount,0
        xor     dl,dl           ;adapter 0
;;;     mov     ax,01h          ;set filter board address
        mov     ax,0ch          ;set filter to promis/bcast
```

```
        @kbdchk                 ;key pressed?
        or      al,al
        jz      rdbfr
        jmp     wedone
rdbfr:
        test    pklock,0ffh     ;got a pkt?
        jnz     lstpkt
        jmp     chkpk
lstpkt:
        test    pkerr,0ffffh    ;any error
        jz      dmpk
        @print  GEmsg
        @prx    4,pkerr
        @print  crlf
        mov     pklock,0
        inc     pkcnt
        jmp     chkpk
dmpk:
        cmp     pklen,0
        jnz     pkok
        @print  ZPmsg
        mov     pklock,0
        inc     pkcnt
        jmp     chkpk
pkok:
        cmp     pklen,256
        jle     dmokl
        mov     pklen,256       ;limit dump to 1st 256 bytes
dmokl:
        @dmprt  <offset CODE:pkthd>,0,pklen
        mov     pklock,0
        inc     pkcnt
        jmp     chkpk

wedone:
        @print  REmsg
        mov     ax,0        ;a return code
        ret

rcvsome endp


; ---------------------------------------------------------------
;       RxProcess
; ---------------------------------------------------------------
RxProcess proc      near

        push    bx
        push    cx

        test    cs:pklock,0ffh
        jz      getp
dontget:
        inc     pkcount
        mov     cx,0        ;zero length (just discard)
        jmp     goget
getp:
        ; At this point we could check es:di packet header data
        ; to make some decision on packet disposition
```

```
        ; lock our buffer and get packet data into it
        mov     cs:pklock,0ffh ;lock buff
        mov     cs:pkerr,0
goget:
        mov     ax,CODE
        mov     es,ax
        mov     di,offset CODE:pkthd     ;buffer
        or      dl,40h      ;release buffer
        ; **********************
        call    GetRxData
        ; **********************
        jcxz    nolen
        mov     cs:pkerr,ax
        mov     cs:pklen,cx

nolen:
        pop     cx
        pop     bx
        ret
RxProcess endp

; ---------------------------------------------------------------
        ExitRcvInt
; ---------------------------------------------------------------
ExitRcvInt  proc    near

        iret

ExitRcvInt  endp

; ---------------------------------------------------------------
; --- get and print WhoAmI statistics ---
; ---------------------------------------------------------------
Dowho   proc    near

        push    es
        xor     dl,dl           ;adapter 0
        ; **********************
        call    WhoAmI
        ; **********************
        mov     retsav,ax

        @print  WAmsg
        @prx    4,retsav
        @print  crlf
        mov     ax,retsav
        or      ax,ax
        jz      wa_ok
        mov     errcd,3
        jmp     uninit

wa_ok:
        mov     si,di
        mov     di,offset CODE:wbf
        push    ds

        push    ds
        push    es
        pop     ds
        pop     es
```

```
        mov     cx,24
        cld
    rep movsw                   ;copy who butter

        pop     ds
        pop     es

        call    whodat          ;print the WhoAmI data

        @print  PAmsg
        mov     ah,8
        int     21h             ;wait for key

        ret
dowho   endp


;-----  print WhoAmI data  ------------------------
whodat  PROC    near
        @print  W00msg

;;;;     @dmprt  <offset CODE:wbf>,0,48

        @print  W01msg
        mov     cx,6
        mov     bx,0
prtea:
        push    bx
        @prx    2,<word ptr [bx+offset CODE:wbf.ea-1]>
        pop     bx
        inc     bx
        loop    prtea
        @print  crlf

        @print  W02msg
        @prx    2,<word ptr wbf.ver1-1>
        @print  crlf

        @print  W03msg
        @prx    2,<word ptr wbf.ver2-1>
        @print  crlf

        @print  W04msg
        @prx    2,<word ptr wbf.ver3-1>
        @print  crlf

        @print  W05msg
        @prx    2,<word ptr wbf.ver4-1>
        @print  crlf

        @print  W06msg
        @prx    2,<word ptr wbf.atyp-1>
        @print  crlf

        @print  W07msg
        @prx    2,<word ptr wbf.astat-1>
        @print  crlf

        @print  W08msg
```

```
        @prx    2,<word ptr wbf.bfrs-1>
        @print  crlf

        @print  W09msg
        @prx    2,<word ptr wbf.nxb-1>
        @print  crlf

        @print  W10msg
        @prx    4,<word ptr wbf.sxb>
        @print  crlf

        @print  W11msg
        @prx    4,<word ptr wbf.xmtc+2>
        @prx    4,<word ptr wbf.xmtc>
        @print  crlf

        @print  W12msg
        @prx    4,<word ptr wbf.xmte+2>
        @prx    4,<word ptr wbf.xmte>
        @print  crlf

        @print  W13msg
        @prx    4,<word ptr wbf.xmtto+2>
        @prx    4,<word ptr wbf.xmtto>
        @print  crlf

        @print  W14msg
        @prx    4,<word ptr wbf.rcvc+2>
        @prx    4,<word ptr wbf.rcvc>
        @print  crlf

        @print  W15msg
        @prx    4,<word ptr wbf.rcvbc+2>
        @prx    4,<word ptr wbf.rcvbc>
        @print  crlf

        @print  W16msg
        @prx    4,<word ptr wbf.rcve+2>
        @prx    4,<word ptr wbf.rcve>
        @print  crlf

        @print  W17msg
        @prx    4,<word ptr wbf.rtc+2>
        @prx    4,<word ptr wbf.rtc>
        @print  crlf

        @print  W18msg
        @prx    2,<word ptr wbf.xfmd-1>
        @print  crlf

        @print  W19msg
        @prx    2,<word ptr wbf.wtmd-1>
        @print  crlf

        @print  W20msg
        @prx    4,<word ptr wbf.extp>
        @print  crlf

        ret
whodat  endp
```

```
;-------------------------------------------------------------------------
savvecs proc    near
        push    ds
        push    es
        push    si
        push    di
        push    cx

        mov     ax,ds
        mov     es,ax
        xor     ax,ax
        mov     ds,ax
        mov     cx,22h*2        ;vectors 0 - 21h, 2 wds per
        mov     di,offset CODE:vectsv
        xor     si,si
        cld
        cli
   rep  movsw                   ;save 'em all
        sti

        pop     cx
        pop     di
        pop     si
        pop     es
        pop     ds
        ret
savvecs endp

;-------------------------------------------------------------------------
fixvecs proc    near
        push    es
        push    si
        push    di
        push    cx

        xor     ax,ax
        mov     es,ax
        mov     cx,22h*2        ;vectors 0 - 21h, 2 wds per
        mov     si,offset CODE:vectsv
        xor     di,di
        cld
        cli
   rep  movsw                   ;restore 'em all
        sti

        pop     cx
        pop     di
        pop     si
        pop     es
        ret
fixvecs endp

;-------------------------------------------------------------------------
;   dmprt - produces dump listing, calling parameters are pushed on stack
;           (converted from a C routine)
;   INPUTS:
;       [bp+4]  = data address
;       [bp+6]  = starting address for line headers
;       [bp+8]  = length of data to print
```

```
;   OUTPUT:
        Dump listing to stdout device
-------------------------------------------------------------------
dmprt   proc    near

        push    bp
        mov     bp,sp
        mov     bx,bp
        sub     bx,0ch          ;local vars
        mov     sp,bx
        push    si
        mov     ax,[bp+8]       ;len

d005c:  sub     dx,dx
        mov     cx,10h

!0061:  div     cx
        mov     [bp-4],ax       ;lines

!0063:  mov     [bp-6],dx       ;rem

d0066:  mov     word ptr [bp-8],0       ;i

d006b:  mov     word ptr [bp-0ah],0     ;line

!0070:  jmp     d0158

d0073:
        push    dx
        mov     dl,cr           ;000d
        mov     ah,2
        int     21h
        mov     dl,lf           ;000A
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

        mov     ax,4
        push    ax
        mov     ax,[bp+6]       ;adr
        add     ax,[bp-8]       ;i
        push    ax
        call    prx
        add     sp,4            ;0004
        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

        mov     word ptr [bp-0ch],0     ;j
```

```
d00eb:  mov     byte ptr [bp-0ch],    ;j
        inc     si
        push    ax
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

d00d5:  mov     ax,2            ;0002
        push    ax
        mov     bx,[bp-8]       ;i
        mov     si,[bp+4]       ;buf
        mov     ah,[bx+si]      ;buf[i]
        push    ax
        call    prx
        add     sp,4            ;0004
        inc     word ptr [bp-8] ;i
        inc     word ptr [bp-0ch]    ;j

d00f0:  cmp     word ptr [bp-0ch],10h    ;j
        jb      d00c5

        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

        sub     word ptr [bp-8],10h     ;i,0010
        mov     word ptr [bp-0ch],0      ;j


        ;do ascii
d0113:  mov     bx,[bp-8]       ;i
        mov     si,[bp+4]       ;buf
        push    dx
        mov     dl,[bx+si]      ;buf[i]
        cmp     dl,' '
        jb      d013f
        cmp     dl,7fh
        jb      d0142

d013f:  mov     dl,'.'          ;002e

d0142:
        mov     ah,2
        int     21h
        pop     dx

        inc     word ptr [bp-8] ;i
        inc     word ptr [bp-0ch]    ;j
        cmp     word ptr [bp-0ch],10h    ;0010
        jb      d0113
        inc     word ptr [bp-0ah]    ;line
```

```
[illegible]     [illegible]
        [illegible]
        jmp     [illegible]
        jmp     d0073


d0163:  cmp     word ptr [bp-6],0       ;rem
        jnz     d016c
        jmp     d0272


d016c:
        push    dx
        mov     dl,cr                   ;000d
        mov     ah,2
        int     21h
        mov     dl,lf                   ;000a
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

        mov     ax,4                    ;0008
        push    ax
        mov     ax,[bp+6]               ;adr
        add     ax,[bp-8]               ;i
        push    ax
        call    prx
        add     sp,4                    ;0004
        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

        mov     word ptr [bp-0ch],0     ;j

        jmp     short   d01c3


d0198:  test    byte ptr [bp-0ch],3     ;j
        jnz     d01a8
        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

d01a8:  mov     ax,2                    ;0002
        push    ax
        mov     bx,[bp-8]               ;i
        mov     si,[bp+4]               ;buf
```

```
        mov     ax,[bp+4]       ;buf,i
        push    ax
        call    prx
        add     sp,4            ;0004
        inc     word ptr [bp-8] ;i
        inc     word ptr [bp-0ch]    ;j

d01c3:  mov     ax,[bp-6]       ;rem
        cmp     [bp-0ch],ax      ;j
        jb      d0198
        jmp     short   d01f4


d01cd:  test    byte ptr [bp-0ch],3  ;j
        jnz     d01dd
        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

d01dd:
        push    dx
        mov     dl,'.'
        mov     ah,2
        int     21h
        mov     dl,'.'
        mov     ah,2
        int     21h
        pop     dx

        inc     word ptr [bp-0ch]    ;j

d01f4:  cmp     word ptr [bp-0ch],10h    ;0010
        jb      d01cd
        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

        mov     ax,[bp-6]       ;rem
        sub     [bp-8],ax       ;i
        mov     word ptr [bp-0ch],0  ;j


        ;do ascii
d0219:  mov     ax,[bp-6]       ;rem
        cmp     [bp-0ch],ax      ;j
        jnb     d026c
        mov     bx,[bp-8]       ;i
        mov     si,[bp+4]       ;buf
        push    dx
        mov     dl,[bx+si]      ;buf[i]
        cmp     dl,' '
        jb      d024d
        cmp     dl,7fh
```

```
        jb      d024d

d024d:  mov     dl,'.'          ;002e

d0250:
        mov     ah,2
        int     21h
        pop     dx

        inc     word ptr [bp-8] ;i
        inc     word ptr [bp-0ch]    ;j
        jmp     short   d0219


d025f:
        push    dx
        mov     dl,'.'
        mov     ah,2
        int     21h
        pop     dx

        inc     word ptr [bp-0ch]    ;j

d026c:  cmp     word ptr [bp-0ch],10h    ;0010
        jb      d025f

d0272:
        push    dx
        mov     dl,cr           ;000d
        mov     ah,2
        int     21h
        mov     dl,1f           ;000a
        mov     ah,2
        int     21h
        pop     dx

        pop     si
        mov     sp,bp
        pop     bp
        ret
dmprt   endp

;-----------------------------------------------------------------------
;   prx - routine to print a hex value from binary data up to word length
;   INPUTS:
;       [bp+4]  = binary data to convert
;       [bp+6]  = number of bytes to print (1 to 4)
;
;-----------------------------------------------------------------------
prx     proc    near

        push    bp
        mov     bp,sp
        mov     bx,bp
        sub     bx,4            ;local space
        mov     sp,bx

        push    si
        push    dx
        push    cx
```

```
        push    ds
        mov     ax,ss           ;make temp buf accessable
        mov     ds,ax
        lea     bx,[bp-4]       ;temp buffer address
        mov     dx,[bp+4]       ;data to cvrt
        call    wtoa
        mov     cx,[bp+6]       ;char count to print
        xor     si,si
prx1:
        mov     dl,[bp+si-4]    ;get a byte
        mov     ah,2
        int     21h             ;print it
        inc     si
        loop    prx1

        pop     ds
        pop     cx
        pop     dx
        pop     si
        mov     sp,bp
        pop     bp
        ret
prx     endp


;---------------------------------------------------------------
;       CONVERT WORD TO ASCII HEX
;    Calling sequence:
;       mov     dx,word         ;word to convert
;       mov     bx,offset out   ;where to put output
;       call    wtoa
;
;       ds:bx  needs 4 bytes for result
;---------------------------------------------------------------

wtoa    proc    near
        push    ax
        push    bx
        push    cx
        push    dx
        push    si
        mov     si,4            ;digits per word
wtoa01:
        mov     al,dl           ;get a digit
        mov     cl,4
        shr     dx,cl           ;strip the digit
        and     al,0fh          ;keep low nibble
        add     al,090h
        daa
        adc     al,040h
        daa
        dec     si              ;count the digit
        mov     [bx+si],al      ;store the digit
        jnz     wtoa01
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        ret
```

```
atoa    endp

RCODE   ENDS
        END tstrx2              ;Ma
```

```
; tstrx4.asm - This program sends packets with random time dela,, data passes
               to the adapter buffer at each packet transfer time.
               It tests the possibilities of overwrite the packet data stayed
;              in the adapter buffer which is delayed to be transfered due to
;              busy channel, collision or abortion.

               ** NOTE: **  To allow this program to end cleanly
;              added savvecs and fixvecs routines to preserve vectors that
               could possibly be changed.
               This allows 3L interrupt hooks to be undone so 3L can be used
;              in an executable program rather than just a permanent driver.


include    ehwie6.h

 define 3L functions
xtrn   InitParameters:near
extrn  InitAdapters:near
xtrn   WhoAmI:near
xtrn   ResetAdapter:near
extrn  RdRxFilter:near
extrn  WrRxFilter:near
xtrn   GetRxData:near
xtrn   SetLookAhead:near
extrn  PutTxData:near

xtrn   SetTime:near
extrn  Ticks:word

xtrn   Srand:near
extrn  Rand:near
extrn  SrandT:near
xtrn   RandT:near
xtrn   Waiting:near
extrn  getpknum:near
xtrn   isxmitok:near
xtrn   stopxmit:near
extrn  getisrtsr:near

xtrn   stop_count :word
extrn  ga_cmd_reg :byte                          ;Ma
extrn  myeaddr    :byte
xtrn   _nxmit     :dword
xtrn   _ntxtmo    :dword
extrn  _ncol      :dword
xtrn   _nmxcol    :dword
xtrn   _nrecv     :dword
extrn  _nbadpk    :dword
extrn  _novflo    :dword
xtrn   _ntxbad    :dword
xtrn   _nrunts    :dword
extrn  _nbrds     :dword
xtrn   _ncolide   :dword                         ;Ma
xtrn   mtoff      :dword                         ;Ma

public  RxProcess
public  ExitRcvInt

; so these'll be in map for debugging
public  argstr, crlf, retsav, pkthd, wbf, xmtpk, fnprmt
```

```
public  xmit, revsome, dowho, savvec, fixvec, dmprt, prx, wtoi

lf      equ     0ah
cr      equ     0dh
minsec  equ     60d

NUMXMIT   equ     10d             ;total packets transmitted      /Ma
WAITIME   equ     16d             ;unit in usec.                  /Ma
RANDRANGE equ     11d             ;upper limit of random number   /Ma
MODUNUM   equ     10d             ;modular number with count      /Ma
FTIME10   equ      1d             ;base time of random time delay /Ma
STOPWAIT  equ      0d             ;1=stop wait, 0=stop no wait    /Ma

$print  macro   strloc          ;print string at strloc
        local   strloc
        push    cx
        lea     dx,strloc
        mov     ah,09h
        int     21h
        pop     cx
        endm

!kbdin  macro                   ;get kbd char in al
        mov     ah,8
        int     21h             ;wait for key
        endm

@kbdchk macro                   ;check for kbd char
        mov     ah,0bh
        int     21h             ;returns al: 0-nokey, ff-keyhit
        endm

!prx    macro   len, dat  .  ;print hex data in word dat, len = 1 to 4
                             ;don't put data in ax
        mov     ax,len
        push    ax
        mov     ax,dat
        push    ax
        call    prx
        add     sp,4
        endm

@dmprt  macro   buf,adr,len         ;hex dump a data area
        mov     ax,len
        push    ax
        mov     ax,adr
        push    ax
        mov     ax,buf
        push    ax
        call    dmprt
        add     sp,6
        endm

CODE    GROUP   DATA, RCODE, STACK

DATA    SEGMENT WORD PUBLIC

;DOS driver init request header format
ini_hd  struc
        db      23      ;hdr len
```

```
        db
        db              ;init cmd
 t.t    dw      0
        db      8 dup (0)
        db      0       ;num units (not used)
 dend   dd      0       ;code end set here
 rgo    dw      0       ;arg offset
 args   dw      0       ;arg segment
        db      0
  ni_hd ends

 ----- adapter parameter setup string --------------------
       this would come from 'device=' on real driver init
argstr  db      "bs.sys  /A:300 /D:1 /I:3",lf

 ---- fake driver init request header for InitParameter input
 h      ini_hd  <,,,,,,,offset CODE:argstr,seg CODE,>

ectsv   dd      22h dup (0)     ;save all vectors so we can cleanup

;WhoAmI adapter info structure
ad_info struc
 a      db      6 dup(0)        ;enet addr
 er1    db      0               ;major ver
ver2    db      0               ;minor ver
 er3    db      0               ;sub ver
 er4    db      0               ;type ver
atyp    db      0               ;adapter type
stat    db      0               ;adapter status
 frs    db      0               ;buffer flags
nxb     db      0               ;number of xmit buffers
sxb     dw      0               ;xmit buffer size
 mtc    dd      0               ;xmit count
 mte    dd      0               ;xmit errs
xmtto   dd      0               ;xmit timeouts
 cvc    dd      0               ;rcv count
 cvbc   dd      0               ;bcast rcv count
rcve    dd      0               ;rcv errs
rtc     dd      0               ;retry count
 fmd    db      0               ;xfer mode flags
wtmd    db      0               ;wait mode flags
extp    dw      0               ;extension pointer
 mtcol  dw      0               ;xmit collision                 /Ma
 d_info ends

program messages
rlf     db      cr,lf,'$'
TVmsg   db      "tst31 load point: $"
IPmsg   db      "InitParameters returns: $"
 Amsg   db      "InitAdapters returns: $"
IAmsg   db      "WhoAmI returns: $"
WFmsg   db      "WrRxFilter returns: $"
 Amsg   db      "SetLookAhead returns: $"
Emsg    db      "GetRxData error return: $"
ZPmsg   db      lf,"Zero length packet",cr,lf,'$'
PAmsg   db      "Press any key to continue",cr,lf,'$'
SSmsg   db      "Starting packet receive... any key to end",cr,lf,'$'
XEmsg   db      "Stopping receive",cr,lf,'$'
CLmsg   db      ":$"
Fmsg    db      " - $"
```

```
[illegible]     db      "Select function, [illegible] for r[illegible]v, [illegible] for x[illegible]: ",'$'
XMmsg           db      "Sending [illegible] packet",cr,lf,'$'
[illegible]Rmsg db      "PutTxData returns: $"

ISRmsg  db      "NICISR value is: $"
TSRmsg  db      "NICTSR value is: $"

RPmsg   db      "Total stop tranmission number: ",'$'           ;Ma
COmsg   db      "Total collision number        : ",'$'          ;Ma
STmsg   db      "Returned TSR decision value   : ",'$'          ;Ma
GAmsg   db      "GA command register value     : ",'$'          ;Ma

XMreq   db      "Transmision of packets has four options:",cr,lf
        db      "  0. Exit",cr,lf
        db      "  1. Generate 78 byte packets randomly w/retry to replace.",cr,
        db      "  2. Generate 142 byte packets randomly w/retry to replace.",cr
        db      "  3. Generate 270 byte packets randomly w/retry to replace.",cr
        db      cr,lf
        db      "Enter your choice: ",'$'                               ;Ma

XMmsg1  db      "Sending 78 bytes packets randomly w/packet replacing."  ;Ma
        db      cr,lf,'$'                                               ;Ma
XMmsg2  db      "Sending 142 bytes packets randomly w/packet replacing." ;Ma
        db      cr,lf,'$'                                               ;Ma
XMmsg3  db      "Sending 270 bytes packets randomly w/packet replacing." ;Ma
        db      cr,lf,'$'                                               ;Ma

W00msg  db      "WhoAmI DATA -",cr,lf,'$'

W01msg  db      "    enet addr              : $"
W02msg  db      "    major ver              : $"
W03msg  db      "    minor ver              : $"
W04msg  db      "    sub ver                : $"
W05msg  db      "    type ver               : $"
W06msg  db      "    adapter type           : $"
W07msg  db      "    adapter status         : $"
W08msg  db      "    buffer flags           : $"
W09msg  db      "    number of xmit buffers : $"
W10msg  db      "    xmit buffer size       : $"
W11msg  db      "    xmit count             : $"
W12msg  db      "    xmit errs              : $"
W13msg  db      "    xmit timeouts          : $"
W14msg  db      "    rcv count              : $"
W15msg  db      "    bcast rcv count        : $"
W16msg  db      "    rcv errs               : $"
W17msg  db      "    retry count            : $"
W18msg  db      "    xfer mode flags        : $"
W19msg  db      "    wait mode flags        : $"
W20msg  db      "    extension pointer      : $"
W21msg  db      "    xmit collision count   : $"          ;Ma

; misc parameters
retsav  dw      ?
segval  dw      ?
toff    dw      ?
errcd   db      0

pklock  db      0
pklen   dw      0
pkerr   dw      0
```

```
pkcnt   dw      0
rkcount dw      0

savax   dw      ?

 receive buffer
pkthd   db      32 dup(0)       ;packet header portion for SetLookAhead
pktdat  db      1500 dup(0)     ;  remainder of pkt buffer

 WhoAmI buffer
wbf     ad_info <>              ;WhoAmI buffer

;***************  ready packet data ************************

;transmit 64 data byte packet
xmtpk   label   byte
desta   db      02h,60h,8ch,01h,02h,03h  ;arbitrary dest addr
sorca   db      00h,00h,00h,0fh,0fh,0fh  ;source addr - fill from who ea
plen    db      0,64                     ;packet length
pdata   db      00h,00h,00h,00h,04h,05h,06h,07h
        db      08h,09h,0ah,0bh,00h,00h,00h,00h
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
        db      20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h,34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh

xplen   dw      $-xmtpk         ;packet len

;***************  ready packet data ************************

;transmit 128 data byte packet
xmtpk2  label   byte
desta2  db      02h,60h,8ch,01h,02h,03h  ;arbitrary dest addr
sorca2  db      00h,00h,00h,0fh,0fh,0fh  ;source addr - fill from who ea
plen2   db      0,128                    ;packet length
pdata2  db      00h,00h,00h,00h,04h,05h,06h,07h
        db      08h,09h,0ah,0bh,00h,00h,00h,00h
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
        db      20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h,34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh
        db      00h,01h,02h,03h,04h,05h,06h,07h
        db      08h,09h,0ah,0bh,0ch,0dh,0eh,0fh
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
        db      20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h,34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh

xplen2  dw      $-xmtpk2        ;packet len

;***************  ready packet data ************************

;transmit 256 data byte packet
xmtpk3  label   byte
```

```
destal  db      0,0h,8ch,01h,02h,03h  ;arbitrary dest addr
sorcal  db      00h,00h,00h,0fh,0fh,0fh  ;source addr - fill from who ea
len3    db      0,255                 ;packet length
pdata3  db      00h,01h,02h,03h,04h,05h,06h,07h
        db      08h,09h,0ah,0bh,00h,00h,00h,00h
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
        db      20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h,34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh
        db      00h,01h,02h,03h,04h,05h,06h,07h
        db      08h,09h,0ah,0bh,0ch,0dh,0eh,0fh
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
        db      20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h,34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh
        db      00h,01h,02h,03h,04h,05h,06h,07h
        db      08h,09h,0ah,0bh,0ch,0dh,0eh,0fh
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
        db      20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h,34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh

plen3   dw      $-xmtpk3        ;packet len

;transmit largest packet, new data area/Ma

;xmtpkl  label   byte
;destal  db      02h,60h,8ch,01h,02h,03h  ;arbitrary dest addr
 sorcal  db      00h,00h,00h,0fh,0fh,0fh  ;source addr - fill from who ea
 plenl   dw      0,1500                   ;packet length
;pdatal  dw      187 dup (0001h,0203h,0405h,0607h,0809h,0a0bh,0c0dh,0e0fh)
        dw      0ff11h,0ff13h

;xplenl  dw      $-xmtpkl        ;packet len

 our     db      0               ;Ma
 in      db      0               ;Ma
sec      db      0               ;Ma
 ount    dw      0               ;Ma
 ountl   dw      0               ;Ma, counts actual packet number
funcnum  db      0               ;Ma
sumrd    dw      0               ;Ma, summation of rand numbers for packet len.
 umrdt   dw      0               ;Ma,     "     "   "      "     " time.
pknum    db      0               ;Ma, packet number 1=78, 2=142, 3=270 bytes
pendflag db      0               ;Ma, 2 - no pending data, 0 - pending data
```

```
DATA    ENDS

STACK   SEGMENT STACK
STACK   ENDS

CODE    SEGMENT WORD PUBLIC
        assume  cs:code, ds:code

;-------------------------------------------------------------------------
;   main routine
;-------------------------------------------------------------------------
tstrx4  proc    near

        mov     ax,CODE
        mov     ds,ax
        mov     es,ax

        mov     ax,cs

        mov     segval,ax
        mov     toff,offset CODE:tst31    ;Ma
        mov     toff,offset CODE:tstrx4   ;Ma

        @print  TVmsg                ;print prog load addr
        @prx    4,segval
        @print  CLmsg
        @prx    4,toff
        @print  crlf
        @print  PAmsg                ;wait for key
        @kbdin                       ; ... get it

        call    savvecs              ;save a bunch of vectors for later

        mov     bx,offset CODE:ih        ;fake driver init request buffer
        ; **********************
        call    InitParameters
        ; **********************
        mov     retsav,ax

        @print  IPmsg
        @prx    4,retsav
        @print  crlf
        mov     ax,retsav
        or      ax,ax
        jz      init_ok
        mov     al,1
        jmp     oout

init_ok:
        mov     di,offset CODE:RxProcess
        ; **********************
        call    InitAdapters
        ; **********************
        mov     retsav,ax

        @print  IAmsg
        @prx    4,retsav
        @print  crlf
        mov     ax,retsav
```

```
        or      ax,ax
        jz      la_ok
        mov     errcd,3
        jmp     uninit

la_ok:

        call    dowho           ;call WhoAmI and list result

        ; SetLookAhead is not required but added for reference
        xor     dl,dl           ;adapter 0
        mov     cx,32           ;LookAhead size
        ; **********************
        call    SetLookAhead
        ; **********************
        mov     retsav,ax

        @print  LAmsg
        @prx    4,retsav
        @print  crlf
        mov     ax,retsav
        or      ax,ax
        jz      la_ok
        mov     errcd,4
        jmp     uninit

la_ok:

        mov     pkcount,0
        xor     dl,dl           ;adapter 0
;;;     mov     ax,01h          ;set filter board address
        mov     ax,0ch          ;set filter to promis/bcast
        ; **********************
        call    WrRxFilter
        ; **********************
        mov     retsav,ax

        @print  WFmsg
        @prx    4,retsav
        @print  crlf
        mov     ax,retsav
        or      ax,ax
        jz      wf_ok
        mov     errcd,5
        jmp     uninit
wf_ok:

;---------------------------------------
        ;do xmit or rcv per user input
fnprmt:
        @print  FNmsg
        @kbdin                  ;get input selection
        push    ax
        @print  crlf
        pop     ax
        cmp     al,'r'
        je      jdorecv         ;Ma
        cmp     al,'t'
        je      doxmt           ;Ma
        jmp     fnprmt          ;Ma
```

```
doxmt:
        mov     stop_count,0    ;Ma, clear # stops

        mov     word ptr _nxmit,0       ;Ma, clear
        mov     word ptr _nxmit+2,0     ;Ma,    _nxmit
        mov     word ptr _nrecv,0       ;Ma, clear
        mov     word ptr _nrecv+2,0     ;Ma,    _nrecv
        mov     word ptr _ncolide,0     ;Ma, clear
        mov     word ptr _ncolide+2,0   ;Ma,    _ncolide

        mov     di,offset CODE:RxProcess  ; reinitialize
        ; **********************
        call    InitAdapters              ;   adapter in a known
        ; **********************
        mov     retsav,ax                 ;       state

        ; SetLookAhead is not required but added for reference
        xor     dl,dl           ;adapter 0
        mov     cx,32           ;LookAhead size
        ; **********************
        call    SetLookAhead
        ; **********************
        mov     retsav,ax

        mov     pkcount,0
        xor     dl,dl           ;adapter 0
;;;     mov     ax,01h          ;set filter board address
        mov     ax,0ch          ;set filter to promis/bcast
        ; **********************
        call    WrRxFilter
        ; **********************
        mov     retsav,ax

        @print  XMreq           ;Ma
        @kbdin                  ;Ma, get input selection
        push    ax              ;Ma
        @print  crlf            ;Ma
        pop     ax              ;Ma
        cmp     al,'1'          ;Ma
        je      jdoxm1          ;Ma, transmit 78 byte packets
        cmp     al,'2'          ;Ma
        je      jdoxm2          ;Ma, transmit 142 byte packets
        cmp     al,'3'          ;Ma
        je      jdoxm3          ;Ma, transmit 270 byte packets
        cmp     al,'0'          ;Ma
        je      juninit         ;Ma, end of transmision
        jmp     doxmt           ;Ma
        mov     errcd,al        ;Ma
        jmp     uninit          ;Ma

juninit:
        jmp     uninit          ;Ma

jdoxm1: jmp     doxm1
jdoxm2: jmp     doxm2
jdoxm3: jmp     doxm3

; ---------------------------------------------------------------------------
```

```
xmit:
        &print  XMmsg1          ;Ma
        mov     funcnum,1d      ;Ma
        mov     count,1d        ;Ma, start count
        mov     count1,1d       ;Ma, start count1
        mov     cx,NUMXMIT
repX1:
        push    cx              ;Ma

        mov     ax,count                ;Ma
        mov     byte ptr pdata[13],ah   ;Ma, mark packet sequential number h
        mov     byte ptr pdata[14],al   ;Ma,      & low bytes
        mov     ax,count1               ;Ma
        mov     byte ptr pdata[15],ah   ;Ma, mark actual packet number on high
        mov     byte ptr pdata[16],al   ;Ma     & low bytes

        call    Xmit1           ;Ma, transmit one "canned" packet

;xmitwait:
        mov     dx,0d           ;Ma
        mov     ax,count        ;Ma
        mov     bx,MODUNUM      ;Ma
        div     bx              ;Ma
        mov     ax,dx           ;Ma, pass seed number to SrandT
        mov     bx,RANDRANGE    ;Ma, pass upper random limit to SrandT
        call    SrandT          ;Ma, set random seed and upper limit numbers
        call    RandT           ;Ma, get random number
        add     sumrdt,ax
        mov     dx,PTIME10      ;Ma, load interframe time
        mul     dx              ;Ma, get total delay time in microseconds
        call    Waiting         ;Ma, delay

        xor     ax,ax               ;Ma, reset ax

        mov     si,WORD PTR mtoff   ;Ma, load ieparams addr.

        mov     dx,IEBASE[si]           ;Ma
        add     dx,NICNCR               ;Ma, get NICNCR address
        in      al,dx                   ;Ma, read collision number
        add     word ptr _ncolide,ax    ;Ma
        adc     word ptr _ncolide+2,0   ;Ma

        call    isxmitok            ;Ma, check transmit status
        cmp     al,1d               ;Ma, returned transmit status value - true
        jz      incount11           ;Ma, trasmission complete
        mov     ax,STOPWAIT         ;Ma, set up wait/no wait flag
        call    stopxmit            ;Ma, stop NIC to transmit
        jmp     short incount1      ;Ma

incount11:
        add     word ptr _nxmit,1       ;Ma, bump counter
        adc     word ptr _nxmit+2,0     ;Ma
        inc     count1                  ;Ma, increament of actual pkt #
        inc     count                   ;Ma, increament of total pkt #
        mov     pendflag,0d             ;Ma, next data - pending
        jmp     short pass1
jrepX1:
        jmp     short repX1
```

```
        inc     count           ;Ma, increment of total pkt #
        mov     pendflag,2d     ;Ma, next d ta - no pending
pass1:
        pop     cx
        dec     cx
        jnz     jrepX1

        call    dowho           ;Ma, list WhoAmI result
        @print  RPmsg           ;
        @prx    4,stop_count
        @print  crlf
        @print  COmsg
        mov     bx,word ptr _ncolide+2
        @prx    4,bx
        @print  crlf
        @print  GAmsg
        mov     bl,ga_cmd_reg
        @prx    4,bx
        @print  crlf
        @print  PAmsg
        mov     ah,8
        int     21h             ;wait for key

        jmp     doxmt           ;Ma,

; ---------------------------------------------------------------------------
; transmit 128 data byte packets w/packet replace.
; ---------------------------------------------------------------------------
doxm2:
        @print  XMmsg2          ;Ma
        mov     funcnum,2d              ;Ma
        mov     count,1d        ;Ma, start count
        mov     count1,1d       ;Ma, start count1
        mov     cx,NUMXMIT
repX2:
        push    cx              ;Ma

        mov     ax,count                ;Ma
        mov     byte ptr pdata2[13],ah  ;Ma, mark packet sequential number h
        mov     byte ptr pdata2[14],al  ;Ma,   & low bytes
        mov     ax,count1               ;Ma
        mov     byte ptr pdata2[15],ah  ;Ma, mark actual packet number high
        mov     byte ptr pdata2[16],al  ;Ma     & low bytes

        call    Xmit1           ;Ma, transmit one "canned" packet

;xmitwait:
        mov     dx,0d           ;Ma
        mov     ax,count        ;Ma
        mov     bx,MODUNUM      ;Ma
        div     bx              ;Ma
        mov     ax,dx           ;Ma, pass seed number to SrandT
        mov     bx,RANDRANGE    ;Ma, pass upper random limit to SrandT
        call    SrandT          ;Ma, set random seed and upper limit numbers
        call    RandT           ;Ma, get random number
        add     sumrdt,ax
        mov     dx,FTIME10      ;Ma, load interframe time
        mul     dx              ;Ma, get total delay time in microseconds
```

```
        call    waiting         ;Ma, delay

        xor     ax,ax                   ; Ma, reset ax

        mov     si,WORD PTR mtoff       ; load ieparams addr.

        mov     dx,IEBASE[si]           ; Ma
        add     dx,NICNCR               ; Ma, get NICNCR address
        in      al,dx                   ; Ma, read collision number
        add     word ptr _ncolide,ax    ; Ma
        adc     word ptr _ncolide+2,0   ; Ma

        call    isxmitok        ;Ma, check transmit status
        cmp     ax,1d           ;Ma, returned transmit status value - true
        jz      incount21       ;Ma, trasmission complete
        mov     ax,STOPWAIT     ;Ma, set up wait/no wait flag
        call    stopxmit        ;Ma, stop NIC to transmit
        jmp     short incount2  ;Ma

incount21:
        add     word ptr _nxmit,1       ;Ma, bump counter
        adc     word ptr _nxmit+2,0     ;Ma
        inc     count1                  ;Ma, increament of actual pkt #
        inc     count                   ;Ma, increament of total pkt #
        jmp     short pass2
incount2:
        inc     count                   ;Ma, increament of total pkt #

pass2:
        pop     cx              ;Ma
        dec     cx              ;Ma
        jnz     jrepX2          ;Ma
        call    dowho           ;Ma, list WhoAmI result
        @print  RPmsg           ;
        @prx    4,stop_count
        @print  crlf
        @print  COmsg
        mov     bx,word ptr _ncolide+2
        @prx    4,bx
        @print  crlf
        jmp     doxmt           ;Ma

jrepX2: jmp     repX2


; ---------------------------------------------------------------------------
; transmit 256 data byte packet w/packet replacing.
; ---------------------------------------------------------------------------
doxm3:
        @print  XMmsg3          ;Ma
        mov     funcnum,3d      ;Ma
        mov     count,1d        ;Ma, start count
        mov     count1,1d       ;Ma, start count1
        mov     cx,NUMXMIT
repX3:
        push    cx              ;Ma

        mov     ax,count                ;Ma
        mov     byte ptr pdata3[13],ah  ;Ma, mark packet sequential number h
        mov     byte ptr pdata3[14],al  ;Ma,     & low bytes
```

```
        mov     ax,count1                       ;Ma
        mov     byte ptr pdata3[15],ah          ;Ma, mark actual packet number on high
        mov     byte ptr pdata3[16],al          ;Ma,      & low bytes

        call    Xmit1           ;Ma, transmit one "canned" packet

.xmitwait:
        mov     dx,0d           ;Ma
        mov     ax,count        ;Ma
        mov     bx,MODUNUM      ;Ma
        div     bx              ;Ma
        mov     ax,dx           ;Ma, pass seed number to SrandT
        mov     bx,RANDRANGE    ;Ma, pass upper random limit to SrandT
        call    SrandT          ;Ma, set random seed and upper limit numbers
        call    RandT           ;Ma, get random number
        add     sumrdt,ax
        mov     dx,FTIME10      ;Ma, load interframe time
        mul     dx              ;Ma, get total delay time in microseconds
        call    Waiting         ;Ma, delay

        xor     ax,ax                   ; Ma, reset ax

        mov     si,WORD PTR mtoff       ; load ieparams addr.

        mov     dx,IEBASE[si]           ; Ma
        add     dx,NICNCR               ; Ma, get NICNCR address
        in      al,dx                   ; Ma, read collision number
        add     word ptr _ncolide,ax    ; Ma
        adc     word ptr _ncolide+2,0   ; Ma

        call    isxmitok        ;Ma, check transmit status
        cmp     ax,1d           ;Ma, returned transmit status value - true
        jz      incount31       ;Ma, trasmission complete
        mov     ax,STOPWAIT     ;Ma, set up wait/no wait flag
        call    stopxmit        ;Ma, stop NIC to transmit
        jmp     short incount3  ;Ma

incount31:
        add     word ptr _nxmit,1       ;Ma, bump counter
        adc     word ptr _nxmit+2,0     ;Ma
        inc     count1                  ;Ma, increament of actual pkt #
        inc     count                   ;Ma, increament of total pkt #
        jmp     short pass3
ncount3:
        inc     count           ;Ma, increament of total pkt #

pass3:
        pop     cx
        dec     cx
        jnz     jrepX3
        call    dowho           ;Ma, list WhoAmI result
        @print  RPmsg           ;
        @prx    4,stop_count
        @print  crlf
        @print  COmsg
        mov     bx,word ptr _ncolide+2
        @prx    4,bx
        @print  crlf
        jmp     doxmt           ;Ma
```

```
jtopX3: jmp     :-topX3

;*****************************************
; receive packets
;*****************************************

dorecv:
        call    rcvsome         ;recieve packets for till key hit
        mov     errcd,al


uninit:
        ; **********************
        call    ResetAdapter
        ; **********************
        call    fixvecs
        mov     al,errcd

bout:   mov     ah,4ch
        int     21h

tstrx4  endp                    ;Ma

; ---------------------------------------------------------------
xmit1   proc    near
; ---------------------------------------------------------------
; transmit one "canned" packet

        ;setup for PutTxData
        cmp     funcnum,3d      ;Ma
        je      set3            ;Ma
        cmp     funcnum,2d      ;Ma
        je      set2            ;Ma
set1:
        ;put our eaddr in xmit pkt
        mov     ax,word ptr wbf.ea
        mov     word ptr sorca,ax
        mov     ax,word ptr wbf.ea+2
        mov     word ptr sorca+2,ax
        mov     ax,word ptr wbf.ea+4
        mov     word ptr sorca+4,ax

        mov     ax,STOPWAIT
        or      al,pendflag
        test    al,3d           ;test pending flag and wait status
        jz      pending         ;pending if STOPWAIT=0 and pendflag=0
        mov     dx,70h          ;req id and no wait, with data pass, no xmit
                                ;count
        jmp     short nopending ;
pending:
        mov     dx,30h          ;req id and no wait, with data pass/pending
                                ;data, no xmit count
nopending:
        mov     si,offset CODE:xmtpk ;xmt pkt buffer
        mov     bx,xplen        ;set lengths
        mov     cx,bx
        jmp     short setnoTx   ;Ma

set2:
        ;put our eaddr in xmit pkt
```

```
        mov     ax,word ptr wbf.ea
        mov     word ptr sorca2,ax
        mov     ax,word ptr wbf.ea+2
        mov     word ptr sorca2+2,ax
        mov     ax,word ptr wbf.ea+4
        mov     word ptr sorca2+4,ax

        mov     dx,70h          ;Ma, req id no wait with data pass, no xmit
                                ;count
        mov     si,offset CODE:xmtpk2 ;xmt pkt buffer
        mov     bx,xplen2       ;set lengths
        mov     cx,bx
        jmp     short setnoTx   ;Ma

set3:
        ;put our eaddr in xmit pkt
        mov     ax,word ptr wbf.ea
        mov     word ptr sorca3,ax
        mov     ax,word ptr wbf.ea+2
        mov     word ptr sorca3+2,ax
        mov     ax,word ptr wbf.ea+4
        mov     word ptr sorca3+4,ax

        mov     dx,70h          ;Ma, req id no wait with data pass, no xmit
                                ;count
        mov     si,offset CODE:xmtpk3 ;xmt pkt buffer
        mov     bx,xplen3       ;set lengths
        mov     cx,bx

setnoTx: mov    di,0ffffh       ;no TxProcess

        ; **********************
        call    PutTxData
        ; **********************
        mov     retsav,ax

;        @print  XRmsg           ;Ma
;        @prx    4,retsav        ;Ma
;        @print  crlf            ;Ma
        mov     ax,retsav
        ret

xmit1   endp

; ---------------------------------------------------------------
rcvsome proc    near
; ---------------------------------------------------------------
; following code to dump received packets for a fixed time
        @print  RSmsg
chkpk:
        @kbdchk                 ;key pressed?
        or      al,al
        jz      rdbfr
        jmp     wedone
rdbfr:
        test    pklock,0ffh     ;got a pkt?
        jnz     lstpkt
        jmp     chkpk
lstpkt:
        test    pkerr,0ffffh    ;any error
```

```
        ; ********************
        jcxz    nolen
        mov     cs:pkerr,ax
        mov     cs:pklen,cx

olen:
        pop     cx
        pop     bx
        ret
xProcess endp

; ---------------------------------------------------------------
        ExitRcvInt
; ---------------------------------------------------------------
ExitRcvInt  proc    near

        iret

ExitRcvInt  endp

; ---------------------------------------------------------------
; --- get and print WhoAmI statistics ---
; ---------------------------------------------------------------
dowho   proc    near

        push    es
        xor     dl,dl           ;adapter 0
        ; **********************
        call    WhoAmI
        ; **********************
        mov     retsav,ax

        @print  WAmsg
        @prx    4,retsav
        @print  crlf
        mov     ax,retsav
        or      ax,ax
        jz      wa_ok
        mov     errcd,3
        jmp     uninit

wa_ok:
        mov     si,di
        mov     di,offset CODE:wbf
        push    ds

        push    ds
        push    es
        pop     ds
        pop     es
        mov     cx,24
        cld
    rep movsw                   ;copy who buffer

        pop     ds
        pop     es

        call    whodat          ;print the WhoAmI data

;       @print  PAmsg
```

```
;       mov     ah,1
        int     21h             ;wait for key

        ret
dowho   endp


;------ print WhoAmI data ------------------------
whodat  PROC    near
        @print  W00msg

;;;;    @dmprt  <offset CODE:wbf>,0,48

        @print  W01msg
        mov     cx,6
        mov     bx,0
prtea:
        push    bx
        @prx    2,<word ptr [bx+offset CODE:wbf.ea-1]>
        pop     bx
        inc     bx
        loop    prtea
        @print  crlf

        @print  W02msg
        @prx    2,<word ptr wbf.ver1-1>
        @print  crlf

        @print  W03msg
        @prx    2,<word ptr wbf.ver2-1>
        @print  crlf

        @print  W04msg
        @prx    2,<word ptr wbf.ver3-1>
        @print  crlf

        @print  W05msg
        @prx    2,<word ptr wbf.ver4-1>
        @print  crlf

        @print  W06msg
        @prx    2,<word ptr wbf.atyp-1>
        @print  crlf

        @print  W07msg
        @prx    2,<word ptr wbf.astat-1>
        @print  crlf

        @print  W08msg
        @prx    2,<word ptr wbf.bfrs-1>
        @print  crlf

        @print  W09msg
        @prx    2,<word ptr wbf.nxb-1>
        @print  crlf

        @print  W10msg
        @prx    4,<word ptr wbf.sxb>
        @print  crlf
```

```
        @print  W11msg
        @prx    4,<word ptr wbf.xmtc+2>
        @prx    4,<word ptr wbf.xmtc>
        @print  crlf

        @print  W12msg
        @prx    4,<word ptr wbf.xmte+2>
        @prx    4,<word ptr wbf.xmte>
        @print  crlf

        @print  W13msg
        @prx    4,<word ptr wbf.xmtto+2>
        @prx    4,<word ptr wbf.xmtto>
        @print  crlf

        @print  W14msg
        @prx    4,<word ptr wbf.rcvc+2>
        @prx    4,<word ptr wbf.rcvc>
        @print  crlf

        @print  W15msg
        @prx    4,<word ptr wbf.rcvbc+2>
        @prx    4,<word ptr wbf.rcvbc>
        @print  crlf

        @print  W16msg
        @prx    4,<word ptr wbf.rcve+2>
        @prx    4,<word ptr wbf.rcve>
        @print  crlf

        @print  W17msg
        @prx    4,<word ptr wbf.rtc+2>
        @prx    4,<word ptr wbf.rtc>
        @print  crlf

        @print  W18msg
        @prx    2,<word ptr wbf.xfmd-1>
        @print  crlf

        @print  W19msg
        @prx    2,<word ptr wbf.wtmd-1>
        @print  crlf

        @print  W20msg
        @prx    4,<word ptr wbf.extp>
        @print  crlf

        @print  W21msg                                          /Ma
        @prx    4,<word ptr wbf.xmtcol>                         /Ma
        @print  crlf                                            /Ma

        ret
vhodat  endp

;-------------------------------------------------------------------------
savvecs proc    near
        push    ds
        push    es
        push    si
```

```
        push    di
        push    cx

        mov     ax,ds
        mov     es,ax
        xor     ax,ax
        mov     ds,ax
        mov     cx,22h*2        ;vectors 0 - 21h, 2 wds per
        mov     di,offset CODE:vectsv
        xor     si,si
        cld
        cli
   rep  movsw                   ;save 'em all
        sti

        pop     cx
        pop     di
        pop     si
        pop     es
        pop     ds
        ret
savvecs endp

;-------------------------------------------------------------------------
fixvecs proc    near
        push    es
        push    si
        push    di
        push    cx

        xor     ax,ax
        mov     es,ax
        mov     cx,22h*2        ;vectors 0 - 21h, 2 wds per
        mov     si,offset CODE:vectsv
        xor     di,di
        cld
        cli
   rep  movsw                   ;restore 'em all
        sti

        pop     cx
        pop     di
        pop     si
        pop     es
        ret
fixvecs endp

;-------------------------------------------------------------------------
;   dmprt - produces dump listing, calling parameters are pushed on stack
;           (converted from a C routine)
;   INPUTS:
;       [bp+4]  = data address
;       [bp+6]  = starting address for line headers
;       [bp+8]  = length of data to print
;   OUTPUT:
;       Dump listing to stdout device
;-------------------------------------------------------------------------
dmprt   proc    near

        push    bp
```

```
        mov     bp,sp
        mov     bx,bp
        sub     bx,0ch          ;local vars
        mov     sp,bx
        push    si
        mov     ax,[bp+8]       ;len

d005c:  sub     dx,dx
        mov     cx,10h

d0061:  div     cx
        mov     [bp-4],ax       ;lines

d0063:  mov     [bp-6],dx       ;rem

d0066:  mov     word ptr [bp-8],0       ;i

d006b:  mov     word ptr [bp-0ah],0     ;line

d0070:  jmp     d0158

d0073:
        push    dx
        mov     dl,cr           ;000d
        mov     ah,2
        int     21h
        mov     dl,1f           ;000A
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

        mov     ax,4
        push    ax
        mov     ax,[bp+6]       ;adr
        add     ax,[bp-8]       ;i
        push    ax
        call    prx
        add     sp,4            ;0004
        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

        mov     word ptr [bp-0ch],0     ;j

d00c5:  test    byte ptr [bp-0ch],3     ;j
```

```
        ;a.     ;
        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

00d5:   mov     ax,2            ;0002
        push    ax
        mov     bx,[bp-8]       ;i
        mov     si,[bp+4]       ;buf
        mov     ah,[bx+si]      ;buf[i]
        push    ax
        call    prx
        add     sp,4            ;0004
        inc     word ptr [bp-8] ;i
        inc     word ptr [bp-0ch]    ;j

00f0:   cmp     word ptr [bp-0ch],10h    ;j
        jb      00c5

        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

        sub     word ptr [bp-8],10h     ;i,0010
        mov     word ptr [bp-0ch],0      ;j


        ;do ascii
0113:   mov     bx,[bp-8]       ;i
        mov     si,[bp+4]       ;buf
        push    dx
        mov     dl,[bx+si]      ;buf[i]
        cmp     dl,' '
        jb      013f
        cmp     dl,7fh
        jb      0142

013f:   mov     dl,'.'          ;002e

0142:
        mov     ah,2
        int     21h
        pop     dx

        inc     word ptr [bp-8] ;i
        inc     word ptr [bp-0ch]    ;j
        cmp     word ptr [bp-0ch],10h   ;0010
        jb      0113
        inc     word ptr [bp-0ah]    ;line

0158:   mov     ax,[bp-4]       ;lines
        cmp     [bp-0ah],ax      ;line
        jnb     0163
```

```
        jmp     d0273

d0163:  cmp     word ptr [bp-6],0       ;rem
        jnz     d016c
        jmp     d0272

d016c:
        push    dx
        mov     dl,cr                   ;000d
        mov     ah,2
        int     21h
        mov     dl,lf                   ;000a
        mov     ah,2
        int     21h
        mov     dl,'='
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

        mov     ax,4                    ;0008
        push    ax
        mov     ax,[bp+6]               ;adr
        add     ax,[bp-8]               ;i
        push    ax
        call    prx
        add     sp,4                    ;0004
        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

        mov     word ptr [bp-0ch],0     ;j

        jmp     short   d01c3

d0198:  test    byte ptr [bp-0ch],3     ;j
        jnz     d01a8
        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

d01a8:  mov     ax,2                    ;0002
        push    ax
        mov     bx,[bp-8]               ;i
        mov     si,[bp+4]               ;buf
        mov     ah,[bx+si]              ;buf[i]
        push    ax
        call    prx
```

```
        ;i
        inc     word ptr [bp-0ch]       ;j

d01c3:  mov     ax,[bp-6]               ;rem
        cmp     [bp-0ch],ax             ;j
        jb      d0198
        jmp     short   d01f4


d01cd:  test    byte ptr [bp-0ch],3     ;j
        jnz     d01dd
        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

d01dd:
        push    dx
        mov     dl,'.'
        mov     ah,2
        int     21h
        mov     dl,'.'
        mov     ah,2
        int     21h
        pop     dx

        inc     word ptr [bp-0ch]       ;j

d01f4:  cmp     word ptr [bp-0ch],10h   ;0010
        jb      d01cd
        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

        mov     ax,[bp-6]               ;rem
        sub     [bp-8],ax               ;i
        mov     word ptr [bp-0ch],0     ;j


        ;do ascii
d0219:  mov     ax,[bp-6]               ;rem
        cmp     [bp-0ch],ax             ;j
        jnb     d026c
        mov     bx,[bp-8]               ;i
        mov     si,[bp+4]               ;buf
        push    dx
        mov     dl,[bx+si]              ;buf[i]
        cmp     dl,' '
        jb      d024d
        cmp     dl,7fh
        jb      d0250

d024d:  mov     dl,'.'                  ;002e
```

```
        ; **********************
        call    InitAdapters
        ; **********************
        mov     retsav,ax

        @print  IAmsg
        @prx    4,retsav
        @print  crlf
        mov     ax,retsav
        or      ax,ax
        jz      ia_ok
        mov     errcd,2
        jmp     uninit

ia_ok:

        call    dowho           ;call WhoAmI and list result

        ; SetLookAhead is not required but added for reference
        xor     dl,dl           ;adapter 0
        mov     cx,32           ;LookAhead size
        ; **********************
        call    SetLookAhead
        ; **********************
        mov     retsav,ax

        @print  LAmsg
        @prx    4,retsav
        @print  crlf
        mov     ax,retsav
        or      ax,ax
        jz      la_ok
        mov     errcd,4
        jmp     uninit

la_ok:

        mov     pkcount,0
        xor     dl,dl           ;adapter 0
;;;     mov     ax,01h          ;set filter board address
        mov     ax,0ch          ;set filter to promis/bcast
        ; **********************
        call    WrRxFilter
        ; **********************
        mov     retsav,ax

        @print  WFmsg
        @prx    4,retsav
        @print  crlf
        mov     ax,retsav
        or      ax,ax
        jz      wf_ok
        mov     errcd,5
        jmp     uninit
wf_ok:

;-------------------------------------------
        ;do xmit or rcv per user input
```

```
        mov     ah,2
        int     21h
        pop     dx

        inc     word ptr [bp-8] ;i
        inc     word ptr [bp-0ch]    ;j
        jmp     short   d0219

d025f:
        push    dx
        mov     dl,'.'
        mov     ah,2
        int     21h
        pop     dx

        inc     word ptr [bp-0ch]    ;j

d026c:  cmp     word ptr [bp-0ch],10h    ;0010
        jb      d025f

d0272:
        push    dx
        mov     dl,cr           ;000d
        mov     ah,2
        int     21h
        mov     dl,lf           ;000a
        mov     ah,2
        int     21h
        pop     dx

        pop     si
        mov     sp,bp
        pop     bp
        ret
dmprt   endp

;-------------------------------------------------------------------------
;   prx - routine to print a hex value from binary data up to word length
;   INPUTS:
;       [bp+4]  = binary data to convert
;       [bp+6]  = number of bytes to print (1 to 4)
;
;-------------------------------------------------------------------------
prx     proc    near

        push    bp
        mov     bp,sp
        mov     bx,bp
        sub     bx,4            ;local space
        mov     sp,bx

        push    si
        push    dx
        push    cx
        push    ds
        mov     ax,ss           ;make temp buf accessable
        mov     ds,ax
```

```
        mov     bx,[bp+4]       ;starting address
        mov     dx,[bp+6]       ;data to print
        call    wtoa
        mov     cx,[bp+6]       ;char count to print
        xor     si,si
prx1:
        mov     dl,[bp+si-4]    ;get a byte
        mov     ah,2
        int     21h             ;print it
        inc     si
        loop    prx1

        pop     ds
        pop     cx
        pop     dx
        pop     si
        mov     sp,bp
        pop     bp
        ret
prx     endp


;-------------------------------------------------------------
;       CONVERT WORD TO ASCII HEX
;    Calling sequence:
        mov     dx,word         ;word to convert
;       mov     bx,offset out   ;where-to put output
;       call    wtoa

        ds:bx  needs 4 bytes for result
;-------------------------------------------------------------

wtoa    proc    near
        push    ax
        push    bx
        push    cx
        push    dx
        push    si
        mov     si,4            ;digits per word
wtoa01:
        mov     al,dl           ;get a digit
        mov     cl,4
        shr     dx,cl           ;strip the digit
        and     al,0fh          ;keep low nibble
        add     al,090h
        daa
        adc     al,040h
        daa
        dec     si              ;count the digit
        mov     [bx+si],al      ;store the digit
        jnz     wtoa01
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        ret
wtoa    endp

RCODE   ENDS
```

```
;txrx.asm - this program sends  packets with even packet number appending
;           on odd number packet. This experiment issues a start transmission
;           command to the adapter after pass the data of odd number packet
;           without wait the completion of transmission. A second start
;           transmission command is issued after the even number packet is
;           passed onto the adapter.
;
;           ** NOTE: **  To allow this program to end cleanly
;           added savvecs and fixvecs routines to preserve vectors that
;           could possibly be changed.
;           This allows 3L interrupt hooks to be undone so 3L can be used
;           in an executable program rather than just a permanent driver.

include    ehwie6.h

 define 3L functions
extrn   InitParameters:near
extrn   InitAdapters:near
extrn   WhoAmI:near
extrn   ResetAdapter:near
extrn   RdRxFilter:near
extrn   WrRxFilter:near
extrn   GetRxData:near
extrn   SetLookAhead:near
extrn   PutTxData:near

extrn   SetTime:near
extrn   Ticks:word

extrn   Srand:near
extrn   Rand:near
extrn   SrandT:near
extrn   RandT:near
extrn   Waiting:near
extrn   getpknum:near
extrn   isxmitok:near
extrn   stopxmit:near
extrn   getisrtsr:near

extrn   stop_count :word                        ;Ma
extrn   ga_cmd_reg :byte                        ;Ma
extrn   fxmitting  :byte
extrn   myeaddr    :byte                        ;Ma
extrn   pendflag   :byte
extrn   _nxmit     :dword
extrn   _ntxtmo    :dword
extrn   _ncol      :dword
extrn   _nmxcol    :dword
extrn   _nrecv     :dword
extrn   _nbadpk    :dword
extrn   _novflo    :dword
extrn   _ntxbad    :dword
extrn   _nrunts    :dword
extrn   _nbrds     :dword                       ;Ma
extrn   _ncolide   :dword                       ;Ma
extrn   mtoff      :dword

public  RxProcess
public  ExitRcvInt
```

```
 so these'll be in zap for debugging
 ublic  argstr, crlf, retsav, pkthd, wbf, xmtpk, inprmt
public  xmitl, rcvsome, dowho, savvecs, fixvecs, dmprt, prx, wtoa
public  dmacount,dmacount1                                          ;Ma

lf      equ     0ah
cr      equ     0dh
 insec  equ     60d

NUMXMIT equ     100d            ;total packets transmitted          /Ma
WAITIME equ      16d            ;unit in usec.                      /Ma
RANDRANGE equ    11d            ;upper limit of random number       /Ma
MODUNUM equ      10d            ;modular number with count          /Ma
PTIME10 equ       1d            ;base time of random time delay     /Ma
STOPWAIT equ      0d            ;1=stop wait, 0=stop no wait        /Ma

@print  macro   strloc          ;print string at strloc
        local   strloc
        push    cx
        lea     dx,strloc
        mov     ah,09h
        int     21h
        pop     cx
        endm

@kbdin  macro                   ;get kbd char in al
        mov     ah,8
        int     21h             ;wait for key
        endm

@kbdchk macro                   ;check for kbd char
        mov     ah,0bh
        int     21h             ;returns al: 0-nokey, ff-keyhit
        endm

@prx    macro   len, dat        ;print hex data in word dat, len = 1 to 4
                                ;don't put data in ax
        mov     ax,len
        push    ax
        mov     ax,dat
        push    ax
        call    prx
        add     sp,4
        endm

@dmprt  macro   buf,adr,len             ;hex dump a data area
        mov     ax,len
        push    ax
        mov     ax,adr
        push    ax
        mov     ax,buf
        push    ax
        call    dmprt
        add     sp,6
        endm

CODE    GROUP   DATA, RCODE, STACK

DATA    SEGMENT WORD PUBLIC
```

```
;;; driver init request header format
ini_hd  struc
        db      23      ;hdr len
        db      0
        db      0       ;init cmd
stat    dw      0
        db      8 dup (0)
        db      0       ;num units (not used)
 dend   dd      0       ;code end set here
argo    dw      0       ;arg offset
 rgs    dw      0       ;arg segment
        db      0
ini_hd  ends

 ---- adapter parameter setup string --------------------
;     this would come from 'device=' on real driver init
argstr  db      "bs.sys  /A:300 /D:1 /I:3",lf

 ---- fake driver init request header for InitParameter input
ih      ini_hd  <,,,,,,,offset CODE:argstr,seg CODE,>

 ectsv  dd      22h dup (0)     ;save all vectors so we can cleanup

;WhoAmI adapter info structure
id_info struc
ea      db      6 dup(0)        ;enet addr
ver1    db      0               ;major ver
ver2    db      0               ;minor ver
ver3    db      0               ;sub ver
ver4    db      0               ;type ver
atyp    db      0               ;adapter type
astat   db      0               ;adapter status
bfrs    db      0               ;buffer flags
nxb     db      0               ;number of xmit buffers
sxb     dw      0               ;xmit buffer size
xmtc    dd      0               ;xmit count
xmte    dd      0               ;xmit errs
xmtto   dd      0               ;xmit timeouts
rcvc    dd      0               ;rcv count
rcvbc   dd      0               ;bcast rcv count
rcve    dd      0               ;rcv errs
rtc     dd      0               ;retry count
xfmd    db      0               ;xfer mode flags
wtmd    db      0               ;wait mode flags
extp    dw      0               ;extension pointer
xmtcol  dw      0               ;xmit collision                 /Ma
id_info ends

;program messages
crlf    db      cr,lf,'$'
TVmsg   db      "tst31 load point: $"
IPmsg   db      "InitParameters returns: $"
IAmsg   db      "InitAdapters returns: $"
WAmsg   db      "WhoAmI returns: $"
WFmsg   db      "WrRxFilter returns: $"
LAmsg   db      "SetLookAhead returns: $"
GEmsg   db      "GetRxData error return: $"
ZPmsg   db      lf,"Zero length packet",cr,lf,'$'
PAmsg   db      "Press any key to continue",cr,lf,'$'
```

```
 Imsg    db      "Starting packet receive... any key to exit",cr,lf,'$'
 Emsg    db      "Stopping receive",cr,lf,'$'
 Lmsg    db      ":$"
HFmsg    db      " - $"
 Nmsg    db      "Select function, r for recv, t for xmit: ",'$'
 Mmsg    db      "Sending 1 packet",cr,lf,'$'
ARmsg    db      "PutTxData returns: $"

 SRmsg   db      "NICISR value is: $"
 SRmsg   db      "NICTSR value is: $"

 Pmsg    db      "Total stop tranmission number: ",'$'          ;Ma
 Omsg    db      "Total collision number       : ",'$'          ;Ma
STmsg    db      "Returned TSR decision value  : ",'$'          ;Ma
GAmsg    db      "GA command register value    : ",'$'          ;Ma

 Mreq    db      "Transmision of packets has four options:",cr,lf
         db      "  0. Exit",cr,lf
         db      "  1. Generate 78 byte packets randomly w/retry to replace.",cr,
         db      "  2. Generate 142 byte packets randomly w/retry to replace.",cr
         db      "  3. Generate 270 byte packets randomly w/retry to replace.",cr
         db      cr,lf
         db      "Enter your choice: ",'$'                                  ;Ma

XMmsg0   db      "Sending 78,142 & 270 bytes packets randomly."             ;Ma
         db      cr,lf,'$'                                                  ;Ma
 Mmsg1   db      "Sending 78 bytes packets randomly w/packet replacing."    ;Ma
         db      cr,lf,'$'                                                  ;Ma
 Mmsg2   db      "Sending 142 bytes packets randomly w/packet replacing."   ;Ma
         db      cr,lf,'$'                                                  ;Ma
XMmsg3   db      "Sending 270 bytes packets randomly w/packet replacing."   ;Ma
         db      cr,lf,'$'                                                  ;Ma

W00msg   db      "WhoAmI DATA -",cr,lf,'$'

 01msg   db      "    enet addr              : $"
 02msg   db      "    major ver              : $"
W03msg   db      "    minor ver              : $"
 04msg   db      "    sub ver                : $"
 05msg   db      "    type ver               : $"
W06msg   db      "    adapter type           : $"
W07msg   db      "    adapter status         : $"
 08msg   db      "    buffer flags           : $"
 09msg   db      "    number of xmit buffers : $"
W10msg   db      "    xmit buffer size       : $"
 11msg   db      "    xmit count             : $"
 12msg   db      "    xmit errs              : $"
W13msg   db      "    xmit timeouts          : $"
 14msg   db      "    rcv count              : $"
 15msg   db      "    bcast rcv count        : $"
w16msg   db      "    rcv errs               : $"
W17msg   db      "    retry count            : $"
 18msg   db      "    xfer mode flags        : $"
 19msg   db      "    wait mode flags        : $"
W20msg   db      "    extension pointer      : $"
W21msg   db      "    xmit collision count   : $"             ;Ma

; misc parameters
retsav   dw      ?
segval   dw      ?
```

```
;off     dw      .
 rcd     db      0

pklock  db      0
pklen   dw      0
 kerr   dw      0
pkcnt   dw      0
pkcount dw      0

 avax   dw      ?

 receive buffer
kthd    db      32 dup(0)       ;packet header portion for SetLookAhead
pktdat  db      1500 dup(0)     ;  remainder of pkt buffer

 WhoAmI buffer
bf      ad_info <>              ;WhoAmI buffer

 ***************  ready packet data *************************

;transmit 64 data byte packet
mtpk    label   byte
esta    db      02h,60h,8ch,01h,02h,03h  ;arbitrary dest addr
orca    db      00h,00h,00h,0fh,0fh,0fh  ;source addr - fill from who ea
len     db      0,64                     ;packet length
data    db      00h,00h,00h,00h,04h,05h,06h,07h
        db      08h,09h,0ah,0bh,00h,00h,00h,00h
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
        db      20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h,34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh

xplen   dw      $-xmtpk         ;packet len

 ***************  ready packet data *************************

;transmit 128 data byte packet
mtpk2   label   byte
desta2  db      02h,60h,8ch,01h,02h,03h  ;arbitrary dest addr
orca2   db      00h,00h,00h,0fh,0fh,0fh  ;source addr - fill from who ea
len2    db      0,128                    ;packet length
data2   db      00h,00h,00h,00h,04h,05h,06h,07h
        db      08h,09h,0ah,0bh,00h,00h,00h,00h
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
        db      20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h,34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh
        db      00h,01h,02h,03h,04h,05h,06h,07h
        db      08h,09h,0ah,0bh,0ch,0dh,0eh,0fh
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
        db      20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h,34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh
```

```
xplen2  dw      $-xmtpk2        ;packet len

 ***************  ready packet data ************************

;transmit 256 data byte packet
;mtpk3  label   byte
desta3  db      02h,60h,8ch,01h,02h,03h  ;arbitrary dest addr
sorca3  db      00h,00h,00h,0fh,0fh,0fh  ;source addr - fill from who ea
plen3   db      0,255                    ;packet length
pdata3  db      00h,01h,02h,03h,04h,05h,06h,07h
        db      08h,09h,0ah,0bh,00h,00h,00h,00h
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
        db      20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h,34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh
        db      00h,01h,02h,03h,04h,05h,06h,07h
        db      08h,09h,0ah,0bh,0ch,0dh,0eh,0fh
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
        db      20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h,34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh
        db      00h,01h,02h,03h,04h,05h,06h,07h
        db      08h,09h,0ah,0bh,0ch,0dh,0eh,0fh
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
        db      20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h,34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh

xplen3  dw      $-xmtpk3        ;packet len

;transmit largest packet, new data area/Ma

;xmtpk1  label   byte
;destal  db      02h,60h,8ch,01h,02h,03h  ;arbitrary dest addr
;sorcal  db      00h,00h,00h,0fh,0fh,0fh  ;source addr - fill from who ea
;plen1   dw      0,1500                   ;packet length
;pdata1  dw      187 dup (0001h,0203h,0405h,0607h,0809h,0a0bh,0c0dh,0e0fh)
;       dw      0ff11h,0ff13h

;xplen1  dw      $-xmtpk1        ;packet len

dmacount  dw    0               ;Ma
dmacount1 dw    0               ;Ma
hour    db      0               ;Ma
min     db      0               ;Ma
sec     db      0               ;Ma
```

```
count    dw       ?                ;Ma
 count1  dw       0                ;Ma, counts actual packet number
 unenum  db       0                ;Ma
sumrd    dw       0                ;Ma, summation of rand numbers for packet len.
sumrdt   dw       0                ;Ma,     "     "   "      "     "  time.
 knum    db       0                ;Ma, packet number 1=78, 2=142, 3=270 bytes

DATA     ENDS

 TACK    SEGMENT STACK
STACK    ENDS


RCODE    SEGMENT WORD PUBLIC
         assume  cs:code, ds:code

;-------------------------------------------------------------------
;   main routine
;-------------------------------------------------------------------
strx6    proc    near

         mov     ax,CODE
         mov     ds,ax
         mov     es,ax

         mov     ax,cs

         mov     segval,ax
          mov     toff,offset CODE:tst31    ;Ma
         mov     toff,offset CODE:tstrx6    ;Ma

         @print  TVmsg                ;print prog load addr
         @prx    4,segval
         @print  CLmsg
         @prx    4,toff
         @print  crlf
         @print  PAmsg                ;wait for key
         @kbdin                       ; ... get it

         call    savvecs              ;save a bunch of vectors for later

         mov     bx,offset CODE:ih        ;fake driver init request buffer
         ; **********************
         call    InitParameters
         ; **********************
         mov     retsav,ax

         @print  IPmsg
         @prx    4,retsav
         @print  crlf
         mov     ax,retsav
         or      ax,ax
         jz      init_ok
         mov     al,1
         jmp     oout

init_ok:
         mov     di,offset CODE:RxProcess
         ; **********************
         call    InitAdapters
```

```
        ; **********************
        mov     retsav,ax

        @print  IAmsg
        @prx    4,retsav
        @print  crlf
        mov     ax,retsav
        or      ax,ax
        jz      ia_ok
        mov     errcd,2
        jmp     uninit

ia_ok:

        call    dowho           ;call WhoAmI and list result

        ; SetLookAhead is not required but added for reference
        xor     dl,dl           ;adapter 0
        mov     cx,32           ;LookAhead size
        ; **********************
        call    SetLookAhead
        ; **********************
        mov     retsav,ax

        @print  LAmsg
        @prx    4,retsav
        @print  crlf
        mov     ax,retsav
        or      ax,ax
        jz      la_ok
        mov     errcd,4
        jmp     uninit

la_ok:

        mov     pkcount,0
        xor     dl,dl           ;adapter 0
;;;     mov     ax,01h          ;set filter board address
        mov     ax,0ch          ;set filter to promis/bcast
        ; **********************
        call    WrRxFilter
        ; **********************
        mov     retsav,ax

        @print  WFmsg
        @prx    4,retsav
        @print  crlf
        mov     ax,retsav
        or      ax,ax
        jz      wf_ok
        mov     errcd,5
        jmp     uninit
wf_ok:

;---------------------------------------
        ;do xmit or rcv per user input
fnprmt:
        @print  FNmsg
        @kbdin                  ;get input selection
        push    ax
```

```
        @print  crlf
        pop     ax
        cmp     al,'r'
        je      jdorecv         ;Ma
        cmp     al,'t'
        je      doxmt           ;Ma
        jmp     fnprmt          ;Ma
jdorecv: jmp    dorecv          ;Ma

doxmt:
        mov     stop_count,0    ;Ma, clear # stops

        mov     word ptr _nxmit,0       ; clear
        mov     word ptr _nxmit+2,0     ;    _nxmit
        mov     word ptr _nrecv,0       ; clear
        mov     word ptr _nrecv+2,0     ;    _nrecv
        mov      word ptr _ncolide,0    ; clear
        mov      word ptr _ncolide+2,0  ;    _ncolide

        @print  XMreq           ;Ma
        @kbdin                  ;Ma, get input selection
        push    ax              ;Ma
        @print  crlf            ;Ma
        pop     ax              ;Ma
        cmp     al,'1'          ;Ma
        je      jdoxm1          ;Ma, transmit 78 byte packets
        cmp     al,'2'          ;Ma
        je      jdoxm2          ;Ma, transmit 142 byte packets
        cmp     al,'3'          ;Ma
        je      jdoxm3          ;Ma, transmit 270 byte packets
        cmp     al,'0'          ;Ma
        je      juninit         ;Ma, end of transmision
        jmp     doxmt           ;Ma
        mov     errcd,al        ;Ma
        jmp     uninit          ;Ma

juninit:
        jmp     uninit          ;Ma

jdoxm1: jmp     doxm1
jdoxm2: jmp     doxm2
jdoxm3: jmp     doxm3

; ---------------------------------------------------------------------------
; transmit 64 data byte packets w/even number packet appending.
; ---------------------------------------------------------------------------
doxm1:
        @print  XMmsg1                  ;Ma
        mov     funcnum,1d              ;Ma
        mov     count,1d                ;Ma, start count
        mov     count1,1d               ;Ma, start count1
        mov     cx,NUMXMIT
repX1:
        push    cx                      ;Ma

        mov     ax,count                ;Ma
        mov     byte ptr pdata[13],ah   ;Ma
        mov     byte ptr pdata[14],al   ;Ma
        mov     ax,count1               ;Ma
        mov     byte ptr pdata[15],ah   ;Ma
```

```
        mov     [illegible] byte ptr [illegible]     ;Ma
        test    count,1d                    ;Ma, test odd number
        jnz     nopend                      ;Ma
        mov     pendflag,1d                 ;Ma, mark even number packet/appending
        jmp     short callXmit1             ;Ma
nopend:
        mov     pendflag,2d                 ;Ma, mark no appending

callXmit1:
        call    Xmit1                       ;Ma, transmit one "canned" packet

        xor     ax,ax                       ;Ma, reset ax

        mov     si,WORD PTR mtoff           ;Ma, load ieparams addr.

        mov     dx,IEBASE[si]               ;Ma
        add     dx,NICNCR                   ;Ma, get NICNCR address
        in      al,dx                       ;Ma, read collision number
        add     word ptr _ncolide,ax        ;Ma
        adc     word ptr _ncolide+2,0       ;Ma


incount11:
        inc     count1                      ;Ma, increament of actual pkt #
        inc     count                       ;Ma, increament of total pkt #
        jmp     short pass1
jrepX1:
        jmp     short repX1

incount1:
;        inc     count                      ;Ma, increament of total pkt #
;        mov     pendflag,2d                ;Ma, next data - no appending
pass1:
        pop     cx
        dec     cx
        jnz     jrepX1

        call    dowho                       ;Ma, list WhoAmI result
        @print  XRmsg
        @prx    4,retsav
        @print  crlf
        @print  RPmsg           ;
        @prx    4,stop_count
        @print  crlf
        @print  COmsg
        mov     bx,word ptr _ncolide+2
        @prx    4,bx
        @print  crlf
        @print  GAmsg
        mov     bl,ga_cmd_reg
        @prx    4,bx
        @print  crlf
        @print  PAmsg
        mov     ah,8
        int     21h             ;wait for key

        jmp     doxmt           ;Ma

; ---------------------------------------------------------------------------
; transmit 128 data byte packets w/packet replace. **** not used
```

```
xmt2:
        @print  XMmsg2          ;Ma
        mov     funcnum,2d              ;Ma
        mov     count,1d        ;Ma, start count
        mov     count1,1d       ;Ma, start count1
        mov     cx,NUMXMIT
repX2:
        push    cx              ;Ma

        mov     ax,count                ;Ma
        mov     byte ptr pdata2[13],ah  ;Ma
        mov     byte ptr pdata2[14],al  ;Ma
        mov     ax,count1               ;Ma
        mov     byte ptr pdata2[15],ah  ;Ma
        mov     byte ptr pdata2[16],al  ;Ma

        call    Xmit1           ;Ma, transmit one "canned" packet

        xor     ax,ax                   ; Ma, reset ax

        mov     si,WORD PTR ntoff       ; load ieparams addr.

        mov     dx,IEBASE[si]           ; Ma
        add     dx,NICNCR               ; Ma, get NICNCR address
        in      al,dx                   ; Ma, read collision number
        add     word ptr _ncolide,ax    ; Ma
        adc     word ptr _ncolide+2,0   ; Ma

        call    isxmitok        ;Ma, check transmit status
        cmp     ax,1d           ;Ma, returned transmit status value - true
        jz      incount21       ;Ma, trasmission complete
        mov     ax,STOPWAIT     ;Ma, set up wait/no wait flag
        call    stopxmit        ;Ma, stop NIC to transmit
        jmp     short incount2  ;Ma

incount21:
        add     word ptr _nxmit,1       ;Ma, bump counter
        adc     word ptr _nxmit+2,0     ;Ma
        inc     count1                  ;Ma, increament of actual pkt #
        inc     count                   ;Ma, increament of total pkt #
        jmp     short pass2
incount2:
        inc     count                   ;Ma, increament of total pkt #

pass2:
        pop     cx
        dec     cx
        jnz     jrepX2
        call    dowho           ;Ma, list WhoAmI result
        @print  RPmsg           ;
        @prx    4,stop_count
        @print  crlf
        @print  COmsg
        mov     bx,word ptr _ncolide+2
        @prx    4,bx
        @print  crlf
        jmp     doxmt           ;Ma

jrepX2: jmp     repX2
```

```
;-------------------------------------------------------------------------
; transmit 256 data byte packet w/packet replacing.  **** not used
;-------------------------------------------------------------------------
loxm3:
        @print  XMmsg3          ;Ma
        mov     funcnum,3d      ;Ma
        mov     count,1d        ;Ma, start count
        mov     count1,1d       ;Ma, start count1
        mov     cx,NUMXMIT
repX3:
        push    cx              ;Ma

        mov     ax,count                    ;Ma
        mov     byte ptr pdata3[13],ah      ;Ma
        mov     byte ptr pdata3[14],al      ;Ma
        mov     ax,count1                   ;Ma
        mov     byte ptr pdata3[15],ah      ;Ma
        mov     byte ptr pdata3[16],al      ;Ma

        call    Xmit1           ;Ma, transmit one "canned" packet

        xor     ax,ax                   ; Ma, reset ax

        mov     si,WORD PTR mtoff       ; load ieparams addr.

        mov     dx,IEBASE[si]           ; Ma
        add     dx,NICNCR               ; Ma, get NICNCR address
        in      al,dx                   ; Ma, read collision number
        add     word ptr _ncolide,ax    ; Ma
        adc     word ptr _ncolide+2,0   ; Ma

        call    isxmitok        ;Ma, check transmit status
        cmp     ax,1d           ;Ma, returned transmit status value - true
        jz      incount31       ;Ma, trasmission complete
        mov     ax,STOPWAIT     ;Ma, set up wait/no wait flag
        call    stopxmit        ;Ma, stop NIC to transmit
        jmp     short incount3  ;Ma

incount31:
        add     word ptr _nxmit,1       ;Ma, bump counter
        adc     word ptr _nxmit+2,0     ;Ma
        inc     count1                  ;Ma, increament of actual pkt #
        inc     count                   ;Ma, increament of total pkt #
        jmp     short pass3
incount3:
        inc     count           ;Ma, increament of total pkt #

pass3:
        pop     cx
        dec     cx
        jnz     jrepX3
        call    dowho           ;Ma, list WhoAmI result
        @print  RPmsg           ;
        @prx    4,stop_count
        @print  crlf
        @print  COmsg
        mov     bx,word ptr _ncolide+2
        @prx    4,bx
```

```
        print  crlf
        jmp     doxmt           ;Ma

jrepX3: jmp     repX3

 ******************************************
, receive packets
;******************************************

lorecv:
        call    rcvsome         ;recieve packets for till key hit
        mov     errcd,al


uninit:
        ; **********************
        call    ResetAdapter
        ; **********************
        call    fixvecs
        mov     al,errcd

>out:   mov     ah,4ch
        int     21h

tstrx6  endp                    ;Ma

. ----------------------------------------------------------------
xmit1   proc    near
  ----------------------------------------------------------------
  transmit one "canned" packet

        ;setup for PutTxData
        cmp     funcnum,3d      ;Ma
        je      set3            ;Ma
        cmp     funcnum,2d      ;Ma
        je      set2            ;Ma
;et1:
        ;put our eaddr in xmit pkt
        mov     ax,word ptr wbf.ea
        mov     word ptr sorca,ax
        mov     ax,word ptr wbf.ea+2
        mov     word ptr sorca+2,ax
        mov     ax,word ptr wbf.ea+4
        mov     word ptr sorca+4,ax

        mov     ax,STOPWAIT
        or      al,pendflag
        test    al,1d           ;test appending flag and wait status
        jnz     pending         ;appending if pendflag=0
        mov     dx,50h          ;req id and with first xfer call/issue xmit

        jmp     short nopending ;
>ending:
        mov     dx,20h          ;req id and wait, with data pass/appending
                                ;data, issue xmit
lopending:
        mov     si,offset CODE:xmtpk ;xmt pkt buffer
        mov     bx,xplen        ;set lengths
        mov     cx,bx
        jmp     short setnoTx   ;Ma
```

```
set2:
        ;put our eaddr in xmit pkt
        mov     ax,word ptr wbf.ea
        mov     word ptr sorca2,ax
        mov     ax,word ptr wbf.ea+2
        mov     word ptr sorca2+2,ax
        mov     ax,word ptr wbf.ea+4
        mov     word ptr sorca2+4,ax

        mov     dx,70h                  ;Ma, req id no wait with data pass, no xmit
                                        ;count
        mov     si,offset CODE:xmtpk2 ;xmt pkt buffer
        mov     bx,xplen2               ;set lengths
        mov     cx,bx
        jmp     short setnoTx           ;Ma

set3:
        ;put our eaddr in xmit pkt
        mov     ax,word ptr wbf.ea
        mov     word ptr sorca3,ax
        mov     ax,word ptr wbf.ea+2
        mov     word ptr sorca3+2,ax
        mov     ax,word ptr wbf.ea+4
        mov     word ptr sorca3+4,ax

        mov     dx,70h                  ;Ma, req id no wait with data pass, no xmit
                                        ;count
        mov     si,offset CODE:xmtpk3 ;xmt pkt buffer
        mov     bx,xplen3               ;set lengths
        mov     cx,bx

setnoTx: mov    di,0ffffh               ;no TxProcess

        ; **********************
        call    PutTxData
        ; **********************
        mov     retsav,ax

;       @print  XRmsg
;       @prx    4,retsav
;       @print  crlf
        mov     ax,retsav
        ret

xmit1   endp

; ---------------------------------------------------------------
rcvsome proc    near
; ---------------------------------------------------------------
; following code to dump received packets for a fixed time
        @print  RSmsg
chkpk:
        @kbdchk
        or      al,al                   ;key pressed?
        jz      rdbfr
        jmp     wedone
rdbfr:
        test    pklock,0ffh             ;got a pkt?
        jnz     lstpkt
```

```
        jmp     chkpk
:tpkt:
        test    pkerr,0ffffh    ;any error
        jz      dmpk
        @print  GEmsg
        @prx    4,pkerr
        @print  crlf
        mov     pklock,0
        inc     pkcnt
        jmp     chkpk
dmpk:
        cmp     pklen,0
        jnz     pkok
        @print  ZPmsg
        mov     pklock,0
        inc     pkcnt
        jmp     chkpk
pkok:
        cmp     pklen,256
        jle     dmokl
        mov     pklen,256       ;limit dump to 1st 256 bytes
dmokl:
        @dmprt  <offset CODE:pkthd>,0,pklen
        mov     pklock,0
        inc     pkcnt
        jmp     chkpk

wedone:
        @print  REmsg
        mov     ax,0        ;a return code
        ret

 cvsome endp


 ---------------------------------------------------------------
        RxProcess
; ---------------------------------------------------------------
RxProcess proc      near

        push    bx
        push    cx

        test    cs:pklock,0ffh
        jz      getp
:ontget:
        inc     pkcount
        mov     cx,0        ;zero length (just discard)
        jmp     goget
!etp:
        ; At this point we could check es:di packet header data
        ; to make some decision on packet disposition

        ; lock our buffer and get packet data into it
        mov     cs:pklock,0ffh ;lock buff
        mov     cs:pkerr,0
!oget:
        mov     ax,CODE
        mov     es,ax
        mov     di,offset CODE:pkthd     ;buffer
```

```
        or      dl,1Ch                  ;release 1 also
        ; **********************
        call    GetRxData
        ; **********************
        jcxz    nolen
        mov     cs:pkerr,ax
        mov     cs:pklen,cx

nolen:
        pop     cx
        pop     bx
        ret
RxProcess endp

; ---------------------------------------------------------------
        ExitRcvInt
; ---------------------------------------------------------------
ExitRcvInt  proc    near

        iret

ExitRcvInt  endp

; ---------------------------------------------------------------
; --- get and print WhoAmI statistics ---
; ---------------------------------------------------------------
dowho   proc    near

        push    es
        xor     dl,dl           ;adapter 0
        ; **********************
        call    WhoAmI
        ; **********************
        mov     retsav,ax

        @print  WAmsg
        @prx    4,retsav
        @print  crlf
        mov     ax,retsav
        or      ax,ax
        jz      wa_ok
        mov     errcd,3
        jmp     uninit

wa_ok:
        mov     si,di
        mov     di,offset CODE:wbf
        push    ds

        push    ds
        push    es
        pop     ds
        pop     es
        mov     cx,24
        cld
    rep movsw                   ;copy who buffer

        pop     ds
        pop     es
```

```
        call    whodat          ;print the WhoAmI data

        @print  PAmsg
;       mov     ah,8
;       int     21h             ;wait for key

        ret
dowho   endp


;-----  print WhoAmI data  ------------------------
whodat  PROC    near
        @print  W00msg

;;;     @dmprt  <offset CODE:wbf>,0,48

        @print  W01msg
        mov     cx,6
        mov     bx,0
prtea:
        push    bx
        @prx    2,<word ptr [bx+offset CODE:wbf.ea-1]>
        pop     bx
        inc     bx
        loop    prtea
        @print  crlf

        @print  W02msg
        @prx    2,<word ptr wbf.ver1-1>
        @print  crlf

        @print  W03msg
        @prx    2,<word ptr wbf.ver2-1>
        @print  crlf

        @print  W04msg
        @prx    2,<word ptr wbf.ver3-1>
        @print  crlf

        @print  W05msg
        @prx    2,<word ptr wbf.ver4-1>
        @print  crlf

        @print  W06msg
        @prx    2,<word ptr wbf.atyp-1>
        @print  crlf

        @print  W07msg
        @prx    2,<word ptr wbf.astat-1>
        @print  crlf

        @print  W08msg
        @prx    2,<word ptr wbf.bfrs-1>
        @print  crlf

        @print  W09msg
        @prx    2,<word ptr wbf.nxb-1>
        @print  crlf
```

```
        @print  W10msg
        @prx    4,<word ptr wbf.sxb>
        @print  crlf

        @print  W11msg
        @prx    4,<word ptr wbf.xmtc+2>
        @prx    4,<word ptr wbf.xmtc>
        @print  crlf

        @print  W12msg
        @prx    4,<word ptr wbf.xmte+2>
        @prx    4,<word ptr wbf.xmte>
        @print  crlf

        @print  W13msg
        @prx    4,<word ptr wbf.xmtto+2>
        @prx    4,<word ptr wbf.xmtto>
        @print  crlf

        @print  W14msg
        @prx    4,<word ptr wbf.rcvc+2>
        @prx    4,<word ptr wbf.rcvc>
        @print  crlf

        @print  W15msg
        @prx    4,<word ptr wbf.rcvbc+2>
        @prx    4,<word ptr wbf.rcvbc>
        @print  crlf

        @print  W16msg
        @prx    4,<word ptr wbf.rcve+2>
        @prx    4,<word ptr wbf.rcve>
        @print  crlf

        @print  W17msg
        @prx    4,<word ptr wbf.rtc+2>
        @prx    4,<word ptr wbf.rtc>
        @print  crlf

        @print  W18msg
        @prx    2,<word ptr wbf.xfmd-1>
        @print  crlf

        @print  W19msg
        @prx    2,<word ptr wbf.wtmd-1>
        @print  crlf

        @print  W20msg
        @prx    4,<word ptr wbf.extp>
        @print  crlf

        @print  W21msg                                          /Ma
        @prx    4,<word ptr wbf.xmtcol>                         /Ma
        @print  crlf                                            /Ma

        ret
hodat   endp

;-----------------------------------------------------------------------
;avvecs proc    near
```

```
        push    ds
        push    es
        push    si
        push    di
        push    cx

        mov     ax,ds
        mov     es,ax
        xor     ax,ax
        mov     ds,ax
        mov     cx,22h*2        ;vectors 0 - 21h, 2 wds per
        mov     di,offset CODE:vectsv
        xor     si,si
        cld
        cli
   rep  movsw                   ;save 'em all
        sti

        pop     cx
        pop     di
        pop     si
        pop     es
        pop     ds
        ret
savvecs endp

;-------------------------------------------------------------------------
fixvecs proc    near
        push    es
        push    si
        push    di
        push    cx

        xor     ax,ax
        mov     es,ax
        mov     cx,22h*2        ;vectors 0 - 21h, 2 wds per
        mov     si,offset CODE:vectsv
        xor     di,di
        cld
        cli
   rep  movsw                   ;restore 'em all
        sti

        pop     cx
        pop     di
        pop     si
        pop     es
        ret
fixvecs endp

;-------------------------------------------------------------------------
;   dmprt - produces dump listing, calling parameters are pushed on stack
;           (converted from a C routine)
;   INPUTS:
;       [bp+4]  = data address
;       [bp+6]  = starting address for line headers
;       [bp+8]  = length of data to print
;   OUTPUT:
;       Dump listing to stdout device
;-------------------------------------------------------------------------
```

```
hapr:   proc    near

        push    bp
        mov     bp,sp
        mov     bx,bp
        sub     bx,0ch          ;local vars
        mov     sp,bx
        push    si
        mov     ax,[bp+8]       ;len

d005c:  sub     dx,dx
        mov     cx,10h

d0061:  div     cx
        mov     [bp-4],ax       ;lines

d0063:  mov     [bp-6],dx       ;rem

d0066:  mov     word ptr [bp-8],0       ;i

d006b:  mov     word ptr [bp-0ah],0     ;line

d0070:  jmp     d0158

d0073:
        push    dx
        mov     dl,cr           ;000d
        mov     ah,2
        int     21h
        mov     dl,1f           ;000A
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

        mov     ax,4
        push    ax
        mov     ax,[bp+6]       ;adr
        add     ax,[bp-8]       ;i
        push    ax
        call    prx
        add     sp,4            ;0004
        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

        mov     word ptr [bp-0ch],0     ;j
```

```
d00c5:  test    byte ptr [bp-0ch],3     ;j
        jnz     d00d5
        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

d00d5:  mov     ax,2            ;0002
        push    ax
        mov     bx,[bp-8]       ;i
        mov     si,[bp+4]       ;buf
        mov     ah,[bx+si]      ;buf[i]
        push    ax
        call    prx
        add     sp,4            ;0004
        inc     word ptr [bp-8] ;i
        inc     word ptr [bp-0ch]    ;j

d00f0:  cmp     word ptr [bp-0ch],10h    ;j
        jb      d00c5

        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

        sub     word ptr [bp-8],10h     ;i,0010
        mov     word ptr [bp-0ch],0      ;j


        ;do ascii
d0113:  mov     bx,[bp-8]       ;i
        mov     si,[bp+4]       ;buf
        push    dx
        mov     dl,[bx+si]      ;buf[i]
        cmp     dl,' '
        jb      d013f
        cmp     dl,7fh
        jb      d0142

d013f:  mov     dl,'.'          ;002e

d0142:
        mov     ah,2
        int     21h
        pop     dx

        inc     word ptr [bp-8] ;i
        inc     word ptr [bp-0ch]    ;j
        cmp     word ptr [bp-0ch],10h    ;0010
        jb      d0113
        inc     word ptr [bp-0ah]    ;line
```

```
d0158:  mov     ax,[bp-4]       ;i
        cmp     [bp-0ch],ax     ;j
        jnb     d0163
        jmp     d0073

d0163:  cmp     word ptr [bp-6],0    ;rem
        jnz     d016c
        jmp     d0272

d016c:
        push    dx
        mov     dl,cr           ;000d
        mov     ah,2
        int     21h
        mov     dl,1f           ;000a
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

        mov     ax,4            ;0008
        push    ax
        mov     ax,[bp+6]       ;adr
        add     ax,[bp-8]       ;i
        push    ax
        call    prx
        add     sp,4            ;0004
        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

        mov     word ptr [bp-0ch],0     ;j

        jmp     short   d01c3

d0198:  test    byte ptr [bp-0ch],3     ;j
        jnz     d01a8
        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

d01a8:  mov     ax,2            ;0002
        push    ax
        mov     bx,[bp-8]       ;i
        mov     si,[bp+4]       ;buf
```

```
        [illegible]
        push    ax
        call    prx
        add     sp,4                ;0004
        inc     word ptr [bp-8] ;i
        inc     word ptr [bp-0ch]    ;j

d01c3:  mov     ax,[bp-6]           ;rem
        cmp     [bp-0ch],ax          ;j
        jb      d0198
        jmp     short   d01f4


d01cd:  test    byte ptr [bp-0ch],3  ;j
        jnz     d01dd
        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

d01dd:
        push    dx
        mov     dl,'.'
        mov     ah,2
        int     21h
        mov     dl,'.'
        mov     ah,2
        int     21h
        pop     dx

        inc     word ptr [bp-0ch]    ;j

d01f4:  cmp     word ptr [bp-0ch],10h    ;0010
        jb      d01cd
        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

        mov     ax,[bp-6]           ;rem
        sub     [bp-8],ax           ;i
        mov     word ptr [bp-0ch],0  ;j


        ;do ascii
d0219:  mov     ax,[bp-6]           ;rem
        cmp     [bp-0ch],ax          ;j
        jnb     d026c
        mov     bx,[bp-8]           ;i
        mov     si,[bp+4]           ;buf
        push    dx
        mov     dl,[bx+si]          ;buf[i]
        cmp     dl,' '
        jb      d024d
        cmp     dl,7fh
```

```
        ;       ...'

d024d:  mov     dl,'.'          ;002e

d0250:
        mov     ah,2
        int     21h
        pop     dx

        inc     word ptr [bp-8] ;i
        inc     word ptr [bp-0ch]    ;j
        jmp     short   d0219


d025f:
        push    dx
        mov     dl,'.'
        mov     ah,2
        int     21h
        pop     dx

        inc     word ptr [bp-0ch]    ;j

d026c:  cmp     word ptr [bp-0ch],10h    ;0010
        jb      d025f

d0272:
        push    dx
        mov     dl,cr           ;000d
        mov     ah,2
        int     21h
        mov     dl,1f           ;000a
        mov     ah,2
        int     21h
        pop     dx

        pop     si
        mov     sp,bp
        pop     bp
        ret
dmprt   endp

;-------------------------------------------------------------------------
;   prx - routine to print a hex value from binary data up to word length
;   INPUTS:
;       [bp+4]  = binary data to convert
;       [bp+6]  = number of bytes to print (1 to 4)
;
;-------------------------------------------------------------------------
prx     proc    near

        push    bp
        mov     bp,sp
        mov     bx,bp
        sub     bx,4            ;local space
        mov     sp,bx

        push    si
        push    dx
        push    cx
```

```
        push    ds
        mov     ax,ss           ;make temp buf addressable
        mov     ds,ax
        lea     bx,[bp-4]       ;temp buffer address
        mov     dx,[bp+4]       ;data to cvrt
        call    wtoa
        mov     cx,[bp+6]       ;char count to print
        xor     si,si
prx1:
        mov     dl,[bp+si-4]    ;get a byte
        mov     ah,2
        int     21h             ;print it
        inc     si
        loop    prx1

        pop     ds
        pop     cx
        pop     dx
        pop     si
        mov     sp,bp
        pop     bp
        ret
prx     endp


;-----------------------------------------------------------
;       CONVERT WORD TO ASCII HEX
;   Calling sequence:
;       mov     dx,word         ;word to convert
;       mov     bx,offset out   ;where to put output
;       call    wtoa
;
;       ds:bx  needs 4 bytes for result
;-----------------------------------------------------------

wtoa    proc    near
        push    ax
        push    bx
        push    cx
        push    dx
        push    si
        mov     si,4            ;digits per word
wtoa01:
        mov     al,dl           ;get a digit
        mov     cl,4
        shr     dx,cl           ;strip the digit
        and     al,0fh          ;keep low nibble
        add     al,090h
        daa
        adc     al,040h
        daa
        dec     si              ;count the digit
        mov     [bx+si],al      ;store the digit
        jnz     wtoa01
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        ret
```

```
exit:   ret
CODE    ENDS
        END tstrx6              ;Ma
```

```
; txtrx7.asm - this program sends packets with even numbered packet passing
;            data onto the space of the odd numbered packet on the adapter.
;            This experiment issues a start tr   mission command to the
;            adapter after pass the data of odu
;            number packet without waiting of the transmission completion.
;            Then, issue a transmission command after the even number packet.
;
;            ** NOTE: **  This program complished with some modifications in
;            the 3L library routine EHWXMIT.ASM

;            ** NOTE: **  To allow this program to end cleanly
;            added savvecs and fixvecs routines to preserve vectors that
;            could possibly be changed.
;            This allows 3L interrupt hooks to be undone so 3L can be used
;            in an executable program rather than just a permanent driver.


include    ehwie6.h

;define 3L functions
extrn   InitParameters:near
extrn   InitAdapters:near
extrn   WhoAmI:near
extrn   ResetAdapter:near
extrn   RdRxFilter:near
extrn   WrRxFilter:near
extrn   GetRxData:near
extrn   SetLookAhead:near
extrn   PutTxData:near

extrn   SetTime:near
extrn   Ticks:word

extrn   Srand:near
extrn   Rand:near
extrn   SrandT:near
extrn   RandT:near
extrn   Waiting:near
extrn   getpknum:near
extrn   isxmitok:near
extrn   stopxmit:near
extrn   getisrtsr:near

extrn   stop_count :word                        ;Ma
extrn   xmit_completel :byte                    ;Ma
extrn   ga_cmd_reg :byte                        ;Ma
extrn   fxmitting  :byte                        ;Ma
extrn   myeaddr    :byte
extrn   pendflag   :byte                        ;Ma
extrn   _nxmit     :dword
extrn   _ntxtmo    :dword
extrn   _ncol      :dword
extrn   _nmxcol    :dword
extrn   _nrecv     :dword
extrn   _nbadpk    :dword
extrn   _novflo    :dword
extrn   _ntxbad    :dword
extrn   _nrunts    :dword
extrn   _nbrds     :dword
extrn   _ncolide   :dword                       ;Ma
```

```
;extrn   mtoff      :dword                          ;Ma

public  RxProcess
public  ExitRcvInt

  so these'll be in map for debugging
public  argstr, crlf, retsav, pkthd, wbf, xmtpk, fnprmt
public  xmit1, rcvsome, dowho, savvecs, fixvecs, dmprt, prx, wtoa
public  dmacount,dmacount1                                  ;Ma

lf      equ     0ah
cr      equ     0dh
minsec  equ     60d

NUMXMIT equ     100d            ;total packets transmitted      /Ma
WAITIME equ     16d             ;unit in usec.                  /Ma
RANDRANGE equ   11d             ;upper limit of random number   /Ma
MODUNUM equ     10d             ;modular number with count      /Ma
BTIME10 equ     1d              ;base time of random time delay /Ma
STOPWAIT equ    0d              ;1=stop wait, 0=stop no wait    /Ma

@print  macro   strloc          ;print string at strloc
        local   strloc
        push    cx
        lea     dx,strloc
        mov     ah,09h
        int     21h
        pop     cx
        endm

@kbdin  macro                   ;get kbd char in al
        mov     ah,8
        int     21h             ;wait for key
        endm

?kbdchk macro                   ;check for kbd char
        mov     ah,0bh
        int     21h             ;returns al: 0-nokey, ff-keyhit
        endm

@prx    macro   len, dat    ;print hex data in word dat, len = 1 to 4
                            ;don't put data in ax
        mov     ax,len
        push    ax
        mov     ax,dat
        push    ax
        call    prx
        add     sp,4
        endm

@dmprt  macro   buf,adr,len         ;hex dump a data area
        mov     ax,len
        push    ax
        mov     ax,adr
        push    ax
        mov     ax,buf
        push    ax
        call    dmprt
        add     sp,6
        endm
```

```
'ODE    GROUP   DATA, RCODE, STACK

DATA    SEGMENT WORD PUBLIC

 DOS driver init request header format
.ni_hd  struc
        db      23      ;hdr len
        db      0
        db      0       ;init cmd
stat    dw      0
        db      8 dup (0)
        db      0       ;num units (not used)
cdend   dd      0       ;code end set here
argo    dw      0       ;arg offset
ırgs    dw      0       ;arg segment
        db      0
ini_hd  ends

 ---- adapter parameter setup string --------------------
;     this would come from 'device=' on real driver init
argstr  db      "bs.sys  /A:300 /D:1 /I:3",lf

,---- fake driver init request header for InitParameter input
ih      ini_hd  <,,,,,,,,offset CODE:argstr,seg CODE,>

rectsv  dd      22h dup (0)     ;save all vectors so we can cleanup

·WhoAmI adapter info structure
ıd_info struc
ea      db      6 dup(0)        ;enet addr
ver1    db      0               ;major ver
'er2    db      0               ;minor ver
rer3    db      0               ;sub ver
ver4    db      0               ;type ver
ıtyp    db      0               ;adapter type
ıstat   db      0               ;adapter status
bfrs    db      0               ;buffer flags
nxb     db      0               ;number of xmit buffers
ɜxb     dw      0               ;xmit buffer size
xmtc    dd      0               ;xmit count
xmte    dd      0               ;xmit errs
cmtto   dd      0               ;xmit timeouts
:cvc    dd      0               ;rcv count
rcvbc   dd      0               ;bcast rcv count
:cve    dd      0               ;rcv errs
:tc     dd      0               ;retry count
xfmd    db      0               ;xfer mode flags
wtmd    db      0               ;wait mode flags
ɜxtp    dw      0               ;extension pointer
.mtcol  dw      0               ;xmit collision                 /Ma
ad_info ends

:program messages
crlf    db      cr,lf,'$'
TVmsg   db      "tst31 load point: $"
IPmsg   db      "InitParameters returns: $"
fAmsg   db      "InitAdapters returns: $"
WAmsg   db      "WhoAmI returns: $"
VFmsg   db      "WrRxFilter returns: $"
```

```
 ?Amsg   db      "GetLookAhead returns: $"
 ?Emsg   db      "GetRxData error return: $"
  Pmsg   db      11,"Zero length packet",cr,lf,'$'
 PAmsg   db      "Press any key to continue",cr,lf,'$'
 RSmsg   db      "Starting packet receive... any key to end",cr,lf,'$'
  Emsg   db      "Stopping receive",cr,lf,'$'
  Lmsg   db      ":$"
 HFmsg   db      " - $"
 TNmsg   db      "Select function, r for recv, t for xmit: ",'$'
  Mmsg   db      "Sending 1 packet",cr,lf,'$'
 xRmsg   db      "PutTxData returns: $"

  SRmsg  db      "NICISR value is: $"
 ?SRmsg  db      "NICTSR value is: $"

 :Pmsg   db      "Total stop tranmission number: ",'$'        ;Ma
 'Omsg   db      "Total collision number       : ",'$'        ;Ma
 STmsg   db      "Returned TSR decision value  : ",'$'        ;Ma
 GAmsg   db      "GA command register value    : ",'$'        ;Ma

 xMreq   db      "Transmision of packets has four options:",cr,lf
         db      "  0. Exit",cr,lf
         db      "  1. Generate 78 byte packets randomly w/retry to replace.",cr,
         db      "  2. Generate 142 byte packets randomly w/retry to replace.",cr
         db      "  3. Generate 270 byte packets randomly w/retry to replace.",cr
         db      cr,lf
         db      "Enter your choice: ",'$'                    ;Ma

 XMmsg0  db      "Sending 78,142 & 270 bytes packets randomly."          ;Ma
         db      cr,lf,'$'                                              ;Ma
 .Mmsg1  db      "Sending 78 bytes packets randomly w/packet replacing." ;Ma
         db      cr,lf,'$'                                              ;Ma
  Mmsg2  db      "Sending 142 bytes packets randomly w/packet replacing." ;Ma
         db      cr,lf,'$'                                              ;Ma
 XMmsg3  db      "Sending 270 bytes packets randomly w/packet replacing." ;Ma
         db      cr,lf,'$'                                              ;Ma

 W00msg  db      "WhoAmI DATA -",cr,lf,'$'

 I01msg  db      "    enet addr              : $"
 .I02msg db      "    major ver              : $"
 W03msg  db      "    minor ver              : $"
 I04msg  db      "    sub ver                : $"
 I05msg  db      "    type ver               : $"
 W06msg  db      "    adapter type           : $"
 W07msg  db      "    adapter status         : $"
 I08msg  db      "    buffer flags           : $"
 W09msg  db      "    number of xmit buffers : $"
 W10msg  db      "    xmit buffer size       : $"
 I11msg  db      "    xmit count             : $"
 I12msg  db      "    xmit errs              : $"
 W13msg  db      "    xmit timeouts          : $"
 ?I14msg db      "    rcv count              : $"
 I15msg  db      "    bcast rcv count        : $"
 W16msg  db      "    rcv errs               : $"
 W17msg  db      "    retry count            : $"
 I18msg  db      "    xfer mode flags        : $"
 .I19msg db      "    wait mode flags        : $"
 W20msg  db      "    extension pointer      : $"
 I21msg  db      "    xmit collision count   : $"          ;Ma
```

```
; misc parameters
oldsav  dw      ?
segval  dw      ?
toff    dw      ?
errcd   db      0

pklock  db      0
pklen   dw      0
pkerr   dw      0
pkcnt   dw      0
pkcount dw      0

savax   dw      ?

;receive buffer
pkthd   db      32 dup(0)       ;packet header portion for SetLookAhead
pktdat  db      1500 dup(0)     ;  remainder of pkt buffer

;WhoAmI buffer
wbf     ad_info <>              ;WhoAmI buffer

;****************  ready packet data *************************

;transmit 64 data byte packet
xmtpk   label   byte
desta   db      02h,60h,8ch,01h,02h,03h  ;arbitrary dest addr
sorca   db      00h,00h,00h,0fh,0fh,0fh  ;source addr - fill from who ea
plen    db      0,64                     ;packet length
pdata   db      00h,00h,00h,00h,04h,05h,06h,07h
        db      08h,09h,0ah,0bh,00h,00h,00h,00h
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
        db      20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h,34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh

xplen   dw      $-xmtpk         ;packet len

;****************  ready packet data *************************

;transmit 128 data byte packet
xmtpk2  label   byte
desta2  db      02h,60h,8ch,01h,02h,03h  ;arbitrary dest addr
sorca2  db      00h,00h,00h,0fh,0fh,0fh  ;source addr - fill from who ea
plen2   db      0,128                    ;packet length
pdata2  db      00h,00h,00h,00h,04h,05h,06h,07h
        db      08h,09h,0ah,0bh,C0h,00h,00h,00h
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
        db      20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h,34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh
        db      00h,01h,02h,03h,04h,05h,06h,07h
        db      08h,09h,0ah,0bh,0ch,0dh,0eh,0fh
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
        db      20h,21h,22h,23h,24h,25h,26h,27h
```

```
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h,34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh

xplen2  dw      $-xmtpk2        ;packet len

 ****************  ready packet data *************************

;transmit 256 data byte packet
xmtpk3  label   byte
desta3  db      02h,60h,8ch,01h,02h,03h  ;arbitrary dest addr
sorca3  db      00h,00h,00h,0fh,0fh,0fh  ;source addr - fill from who ea
plen3   db      0,255                    ;packet length
pdata3  db      00h,01h,02h,03h,04h,05h,06h,07h
        db      08h,09h,0ah,0bh,00h,00h,00h,00h
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
        db      20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h,34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh
        db      00h,01h,02h,03h,04h,05h,06h,07h
        db      08h,09h,0ah,0bh,0ch,0dh,0eh,0fh
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
        db      20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h,34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh
        db      00h,01h,02h,03h,04h,05h,06h,07h
        db      08h,09h,0ah,0bh,0ch,0dh,0eh,0fh
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
        db      20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h,34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh
        db      00h,01h,02h,03h,04h,05h,06h,07h
        db      08h,09h,0ah,0bh,0ch,0dh,0eh,0fh
        db      10h,11h,12h,13h,14h,15h,16h,17h
        db      18h,19h,1ah,1bh,1ch,1dh,1eh,1fh
        db      20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh
        db      30h,31h,32h,33h,34h,35h,36h,37h
        db      38h,39h,3ah,3bh,3ch,3dh,3eh,3fh

xplen3  dw      $-xmtpk3        ;packet len

;transmit largest packet, new data area/Ma

;xmtpk1  label   byte
;desta1  db      02h,60h,8ch,01h,02h,03h  ;arbitrary dest addr
;sorca1  db      00h,00h,00h,0fh,0fh,0fh  ;source addr - fill from who ea
;plen1   dw      0,1500                   ;packet length
;pdata1  dw      187 dup (0001h,0203h,0405h,0607h,0809h,0a0bh,0c0dh,0e0fh)
;        dw      0ff11h,0ff13h

;xplen1  dw      $-xmtpk1        ;packet len

dmacount  dw     0               ;Ma
```

```
dircount1 dw      0                 ;Ma
hour      db      0                 ;Ma
min       db      0                 ;Ma
sec       db      0                 ;Ma
count     dw      0                 ;Ma
count1    dw      0                 ;Ma, counts actual packet number
funcnum   db      0                 ;Ma
sumrd     dw      0                 ;Ma, summation of rand numbers for packet len.
sumrdt    dw      0                 ;Ma,     "     "   "      "     "  time.
pknum     db      0                 ;Ma, packet number 1=78, 2=142, 3=270 bytes

DATA    ENDS

STACK   SEGMENT STACK
STACK   ENDS


RCODE   SEGMENT WORD PUBLIC
        assume  cs:code, ds:code

;-----------------------------------------------------------------
;   main routine
;-----------------------------------------------------------------
tstrx7  proc    near

        mov     ax,CODE
        mov     ds,ax
        mov     es,ax

        mov     ax,cs

        mov     segval,ax
;       mov     toff,offset CODE:tst31    ;Ma
        mov     toff,offset CODE:tstrx7   ;Ma

        @print  TVmsg                 ;print prog load addr
        @prx    4,segval
        @print  CLmsg
        @prx    4,toff
        @print  crlf
        @print  PAmsg                 ;wait for key
        @kbdin                        ; ... get it

        call    savvecs               ;save a bunch of vectors for later

        mov     bx,offset CODE:ih         ;fake driver init request buffer
        ; **********************
        call    InitParameters
        ; **********************
        mov     retsav,ax

        @print  IPmsg
        @prx    4,retsav
        @print  crlf
        mov     ax,retsav
        or      ax,ax
        jz      init_ok
        mov     al,1
        jmp     oout
```

```
fnprmt:
        @print  FNmsg
        @kbdin                  ;get input selection
        push    ax
        @print  crlf
        pop     ax
        cmp     al,'r'
        je      jdorecv         ;Ma
        cmp     al,'t'
        je      doxmt           ;Ma
        jmp     fnprmt          ;Ma
jdorecv: jmp    dorecv          ;Ma

doxmt:
        mov     stop_count,0    ;Ma, clear # stops

        mov     word ptr _nxmit,0       ; clear
        mov     word ptr _nxmit+2,0     ;    _nxmit
        mov     word ptr _nrecv,0       ; clear
        mov     word ptr _nrecv+2,0     ;    _nrecv
        mov      word ptr _ncolide,0    ; clear
        mov      word ptr _ncolide+2,0  ;    _ncolide

        @print  XMreq           ;Ma
        @kbdin                  ;Ma, get input selection
        push    ax              ;Ma
        @print  crlf            ;Ma
        pop     ax              ;Ma
        cmp     al,'1'          ;Ma
        je      jdoxm1          ;Ma, transmit 78 byte packets
        cmp     al,'2'          ;Ma
        je      jdoxm2          ;Ma, transmit 142 byte packets
        cmp     al,'3'          ;Ma
        je      jdoxm3          ;Ma, transmit 270 byte packets
        cmp     al,'0'          ;Ma
        je      juninit         ;Ma, end of transmision
        jmp     doxmt           ;Ma
        mov     errcd,al        ;Ma
        jmp     uninit          ;Ma

juninit:
        jmp     uninit          ;Ma

jdoxm1: jmp     doxm1
jdoxm2: jmp     doxm2
jdoxm3: jmp     doxm3

; ---------------------------------------------------------------------------
; transmit 64 data byte packets w/even number packet replacing.
; ---------------------------------------------------------------------------
doxm1:
        @print  XMmsg1                  ;Ma
        mov     funcnum,1d              ;Ma
        mov     count,1d                ;Ma, start count
        mov     count1,1d               ;Ma, start count1
        mov     cx,NUMXMIT
repX1:
        push    cx                      ;Ma

        mov     ax,count                ;Ma
```

```
        mov     byte ptr pdata[13],ah   ;Ma
        mov     byte ptr pdata[14],al   ;Ma
        mov     ax,count1               ;Ma
        mov     byte ptr pdata[15],ah   ;Ma
        mov     byte ptr pdata[16],al   ;Ma
        test    count,1d                ;Ma, test odd number
        jnz     nopend                  ;Ma
        mov     pendflag,1d             ;Ma, mark no appending/xmit double
                                        ;Ma,      size
        jmp     short callXmit1         ;Ma
nopend:
        mov     pendflag,2d             ;Ma, mark no appending

callXmit1:
        call    Xmit1                   ;Ma, transmit one "canned" packet

        xor     ax,ax                   ;Ma, reset ax

        mov     si,WORD PTR mtoff       ;Ma, load ieparams addr.

        mov     dx,IEBASE[si]           ;Ma
        add     dx,NICNCR               ;Ma, get NICNCR address
        in      al,dx                   ;Ma, read collision number
        add     word ptr _ncolide,ax    ;Ma
        adc     word ptr _ncolide+2,0   ;Ma


incount11:
        add     word ptr _nxmit,1       ;Ma, bump counter
        adc     word ptr _nxmit+2,0     ;Ma
        inc     count1                  ;Ma, increament of actual pkt #
        inc     count                   ;Ma, increament of total pkt #
        mov     pendflag,0d             ;Ma, next data - appending
        jmp     short pass1
jrepX1:
        jmp     short repX1

incount1:
;       inc     count           ;Ma, increament of total pkt #
;       mov     pendflag,2d     ;Ma, next data - no appending
pass1:
        pop     cx
        dec     cx
        jnz     jrepX1

        call    dowho           ;Ma, list WhoAmI result
        @print  XRmsg
        @prx    4,retsav
        @print  crlf
        mov     al,xmit_complete1       ;Ma
        mov     stop_count,ax           ;Ma
        @print  RPmsg           ;
        @prx    4,stop_count
        @print  crlf
        @print  COmsg
        mov     bx,word ptr _ncolide+2
        @prx    4,bx
        @print  crlf
        @print  GAmsg
        mov     bl,ga_cmd_reg
```

```
        @prx    1,bx
        @print  crlf
        @print  PAmsg
        mov     ah,8
        int     21h             ;wait for key

        jmp     doxmt           ;Ma

; ---------------------------------------------------------------------------
; transmit 128 data byte packets w/packet replace. ******* not used
; ---------------------------------------------------------------------------
doxm2:
        @print  XMmsg2          ;Ma
        mov     funcnum,2d              ;Ma
        mov     count,1d        ;Ma, start count
        mov     count1,1d       ;Ma, start count1
        mov     cx,NUMXMIT
repX2:
        push    cx              ;Ma

        mov     ax,count                ;Ma
        mov     byte ptr pdata2[13],ah  ;Ma
        mov     byte ptr pdata2[14],al  ;Ma
        mov     ax,count1               ;Ma
        mov     byte ptr pdata2[15],ah  ;Ma
        mov     byte ptr pdata2[16],al  ;Ma

        call    Xmit1           ;Ma, transmit one "canned" packet

:xmitwait:
        mov     dx,0d           ;Ma
        mov     ax,count        ;Ma
        mov     bx,MODUNUM      ;Ma
        div     bx              ;Ma
        mov     ax,dx           ;Ma, pass seed number to SrandT
        mov     bx,RANDRANGE    ;Ma, pass upper random limit to SrandT
        call    SrandT          ;Ma, set random seed and upper limit numbers
        call    RandT           ;Ma, get random number
        add     sumrdt,ax
        mov     dx,FTIME10      ;Ma, load interframe time
        mul     dx              ;Ma, get total delay time in microseconds
        call    Waiting         ;Ma, delay

        xor     ax,ax                   ; Ma, reset ax

        mov     si,WORD PTR mtoff       ; load ieparams addr.

        mov     dx,IEBASE[si]           ; Ma
        add     dx,NICNCR               ; Ma, get NICNCR address
        in      al,dx                   ; Ma, read collision number
        add     word ptr _ncolide,ax    ; Ma
        adc     word ptr _ncolide+2,0   ; Ma

        call    isxmitok        ;Ma, check transmit status
        cmp     ax,1d           ;Ma, returned transmit status value - true
        jz      incount21       ;Ma, trasmission complete
        mov     ax,STOPWAIT     ;Ma, set up wait/no wait flag
        call    stopxmit        ;Ma, stop NIC to transmit
        jmp     short incount2  ;Ma
```

```
[illegible]:
        add     word ptr _nxmit,1       ;Ma, [illegible]
        adc     word ptr _nxmit+2,0     ;Ma
        inc     count1                  ;Ma, increament of actual pkt #
        inc     count                   ;Ma, increament of total pkt #
        jmp     short pass2
incount2:
        inc     count                   ;Ma, increament of total pkt #

pass2:
        pop     cx
        dec     cx
        jnz     jrepX2

        call    dowho           ;Ma, list WhoAmI result
        @print  RPmsg           ;
        @prx    4,stop_count
        @print  crlf
        @print  COmsg
        mov     bx,word ptr _ncolide+2
        @prx    4,bx
        @print  crlf
        jmp     doxmt           ;Ma

jrepX2: jmp     repX2


; ---------------------------------------------------------------------
: transmit 256 data byte packet w/packet replacing. **** not used
: ---------------------------------------------------------------------
doxm3:
        @print  XMmsg3          ;Ma
        mov     funcnum,3d      ;Ma
        mov     count,1d        ;Ma, start count
        mov     count1,1d       ;Ma, start count1
        mov     cx,NUMXMIT
repX3:
        push    cx              ;Ma

        mov     ax,count                ;Ma
        mov     byte ptr pdata3[13],ah  ;Ma
        mov     byte ptr pdata3[14],al  ;Ma
        mov     ax,count1               ;Ma
        mov     byte ptr pdata3[15],ah  ;Ma
        mov     byte ptr pdata3[16],al  ;Ma

        call    Xmit1           ;Ma, transmit one "canned" packet

;xmitwait:
        mov     dx,0d           ;Ma
        mov     ax,count        ;Ma
        mov     bx,MODUNUM      ;Ma
        div     bx              ;Ma
        mov     ax,dx           ;Ma, pass seed number to SrandT
        mov     bx,RANDRANGE    ;Ma, pass upper random limit to SrandT
        call    SrandT          ;Ma, set random seed and upper limit numbers
        call    RandT           ;Ma, get random number
        add     sumrdt,ax
        mov     dx,FTIME10      ;Ma, load interframe time
        mul     dx              ;Ma, get total delay time in microseconds
```

```
        call    [illegible]             ;Ma, [illegible]

        xor     ax,ax                   ; Ma, reset ax

        mov     si,WORD PTR mtoff       ; load ieparams addr.

        mov     dx,IEBASE[si]           ; Ma
        add     dx,NICNCR               ; Ma, get NICNCR address
        in      al,dx                   ; Ma, read collision number
        add     word ptr _ncolide,ax    ; Ma
        adc     word ptr _ncolide+2,0   ; Ma

        call    isxmitok        ;Ma, check transmit status
        cmp     ax,1d           ;Ma, returned transmit status value - true
        jz      incount31       ;Ma, trasmission complete
        mov     ax,STOPWAIT     ;Ma, set up wait/no wait flag
        call    stopxmit        ;Ma, stop NIC to transmit
        jmp     short incount3  ;Ma

incount31:
        add     word ptr _nxmit,1       ;Ma, bump counter
        adc     word ptr _nxmit+2,0     ;Ma
        inc     count1                  ;Ma, increament of actual pkt #
        inc     count                   ;Ma, increament of total pkt #
        jmp     short pass3
incount3:
        inc     count           ;Ma, increament of total pkt #

pass3:
        pop     cx
        dec     cx
        jnz     jrepX3

        call    dowho           ;Ma, list WhoAmI result
        @print  RPmsg           ;
        @prx    4,stop_count
        @print  crlf
        @print  COmsg
        mov     bx,word ptr _ncolide+2
        @prx    4,bx
        @print  crlf
        jmp     doxmt           ;Ma

jrepX3: jmp     repX3

;*****************************************
; receive packets
;*****************************************

dorecv:
        call    rcvsome         ;recieve packets for till key hit
        mov     errcd,al


uninit:
        ; **********************
        call    ResetAdapter
        ; **********************
        call    fixvecs
        mov     al,errcd
```

```
xout:   mov     ah,4ch
        int     21h

tstrx7  endp                    ;Ma

; ---------------------------------------------------------------
xmit1   proc    near
; ---------------------------------------------------------------
; transmit one "canned" packet

        ;setup for PutTxData
        cmp     funcnum,3d      ;Ma
        je      set3            ;Ma
        cmp     funcnum,2d      ;Ma
        je      set2            ;Ma
set1:
        ;put our eaddr in xmit pkt
        mov     ax,word ptr wbf.ea
        mov     word ptr sorca,ax
        mov     ax,word ptr wbf.ea+2
        mov     word ptr sorca+2,ax
        mov     ax,word ptr wbf.ea+4
        mov     word ptr sorca+4,ax

        mov     al,pendflag     ;Ma
        test    al,2d           ;Ma, test appending flag and wait status
        jz      pending         ;Ma, appending if pendflag=0
        mov     dx,50h          ;Ma, req id and with first xfer call, issue
                                ;Ma, xmit/no wait
        jmp     short nopending ;
pending:
        mov     dx,60h          ;Ma, req id and wait, with data pass/no appendi
                                ;Ma, data, issue xmit/wait
nopending:
        mov     si,offset CODE:xmtpk ;xmt pkt buffer
        mov     bx,xplen        ;set lengths
        mov     cx,bx
        jmp     short setnoTx   ;Ma

set2:
        ;put our eaddr in xmit pkt
        mov     ax,word ptr wbf.ea      ;Ma
        mov     word ptr sorca2,ax      ;Ma
        mov     ax,word ptr wbf.ea+2    ;Ma
        mov     word ptr sorca2+2,ax    ;Ma
        mov     ax,word ptr wbf.ea+4    ;Ma
        mov     word ptr sorca2+4,ax    ;Ma

        mov     dx,70h          ;Ma, req id no wait with data pass, no xmit
                                ;count
        mov     si,offset CODE:xmtpk2 ;xmt pkt buffer
        mov     bx,xplen2       ;set lengths
        mov     cx,bx
        jmp     short setnoTx   ;Ma

set3:
        ;put our eaddr in xmit pkt
        mov     ax,word ptr wbf.ea
        mov     word ptr sorca3,ax
```

```
        mov     ax,word ptr wbf.ca+2
        mov     word ptr sorca3+2,ax
        mov     ax,word ptr wbf.ca+4
        mov     word ptr sorca3+4,ax

        mov     dx,70h          ;Ma, req id no wait with data pass, no xmit
                                ;count
        mov     si,offset CODE:xmtpk3 ;xmt pkt buffer
        mov     bx,xplen3       ;set lengths
        mov     cx,bx

retnoTx: mov    di,0ffffh       ;no TxProcess

        ; **********************
        call    PutTxData
        ; **********************
        mov     retsav,ax

         @print  XRmsg                      ;/Ma
         @prx    4,retsav                   ;/Ma
;        @print  crlf                       ;/Ma
        mov     ax,retsav
        ret

xmit1   endp

; ---------------------------------------------------------------
rcvsome proc    near
; ---------------------------------------------------------------
; following code to dump received packets for a fixed time
        @print  RSmsg
chkpk:
        @kbdchk                 ;key pressed?
        or      al,al
        jz      rdbfr
        jmp     wedone
rdbfr:
        test    pklock,0ffh     ;got a pkt?
        jnz     lstpkt
        jmp     chkpk
lstpkt:
        test    pkerr,0ffffh    ;any error
        jz      dmpk
        @print  GEmsg
        @prx    4,pkerr
        @print  crlf
        mov     pklock,0
        inc     pkcnt
        jmp     chkpk
dmpk:
        cmp     pklen,0
        jnz     pkok
        @print  ZPmsg
        mov     pklock,0
        inc     pkcnt
        jmp     chkpk
pkok:
        cmp     pklen,256
        jle     dmok1
        mov     pklen,256       ;limit dump to 1st 256 bytes
```

```
1
lnok1:
        @dmprt   offset CODE:pkthd,0,pklen
        mov     pklock,0
        inc     pkcnt
        jmp     chkpk

wedone:
        @print  REmsg
        mov     ax,0        ;a return code
        ret

rcvsome endp


; ---------------------------------------------------------------
;       RxProcess
; ---------------------------------------------------------------
RxProcess proc      near

        push    bx
        push    cx

        test    cs:pklock,0ffh
        jz      getp
dontget:
        inc     pkcount
        mov     cx,0        ;zero length (just discard)
        jmp     goget
getp:
        ; At this point we could check es:di packet header data
        ; to make some decision on packet disposition

        ; lock our buffer and get packet data into it
        mov     cs:pklock,0ffh ;lock buff
        mov     cs:pkerr,0
goget:
        mov     ax,CODE
        mov     es,ax
        mov     di,offset CODE:pkthd     ;buffer
        or      dl,40h                   ;release buffer
        ; **********************
        call    GetRxData
        ; **********************
        jcxz    nolen
        mov     cs:pkerr,ax
        mov     cs:pklen,cx

nolen:
        pop     cx
        pop     bx
        ret
RxProcess endp

; ---------------------------------------------------------------
;       ExitRcvInt
; ---------------------------------------------------------------
ExitRcvInt  proc    near

        iret
```

```
;xitRcvInt  endp

; ---------------------------------------------------------------
; --- get and print WhoAmI statistics ---
; ---------------------------------------------------------------
dowho   proc    near

        push    es
        xor     dl,dl -         ;adapter 0
        ; **********************
        call    WhoAmI
        ; **********************
        mov     retsav,ax

        @print  WAmsg
        @prx    4,retsav
        @print  crlf
        mov     ax,retsav
        or      ax,ax
        jz      wa_ok
        mov     errcd,3
        jmp     uninit

wa_ok:
        mov     si,di
        mov     di,offset CODE:wbf
        push    ds
                "
        push    ds
        push    es
        pop     ds
        pop     es
        mov     cx,24
        cld
    rep movsw                   ;copy who buffer

        pop     ds
        pop     es

        call    whodat          ;print the WhoAmI data

;        @print  PAmsg
;        mov     ah,8
;        int     21h             ;wait for key

        ret
dowho   endp


;-----  print WhoAmI data  ------------------------
whodat  PROC    near
        @print  W00msg

;;;;     @dmprt  <offset CODE:wbf>,0,48

        @print  W01msg
        mov     cx,6
        mov     bx,0
prtea:
```

```
push    bx
@prx    2,<word ptr [bx+offset wbf.wbf1-1]
pop     bx
inc     bx
loop    prtea
@print  crlf

@print  W02msg
@prx    2,<word ptr wbf.ver1-1>
@print  crlf

@print  W03msg
@prx    2,<word ptr wbf.ver2-1>
@print  crlf

@print  W04msg
@prx    2,<word ptr wbf.ver3-1>
@print  crlf

@print  W05msg
@prx    2,<word ptr wbf.ver4-1>
@print  crlf

@print  W06msg
@prx    2,<word ptr wbf.atyp-1>
@print  crlf

@print  W07msg
@prx    2,<word ptr wbf.astat-1>
@print  crlf

@print  W08msg
@prx    2,<word ptr wbf.bfrs-1>
@print  crlf

@print  W09msg
@prx    2,<word ptr wbf.nxb-1>
@print  crlf

@print  W10msg
@prx    4,<word ptr wbf.sxb>
@print  crlf

@print  W11msg
@prx    4,<word ptr wbf.xmtc+2>
@prx    4,<word ptr wbf.xmtc>
@print  crlf

@print  W12msg
@prx    4,<word ptr wbf.xmte+2>
@prx    4,<word ptr wbf.xmte>
@print  crlf

@print  W13msg
@prx    4,<word ptr wbf.xmtto+2>
@prx    4,<word ptr wbf.xmtto>
@print  crlf

@print  W14msg
@prx    4,<word ptr wbf.rcvc+2>
```

```
        @prx    4,<word ptr wbf.rcvc
        @print  crlf

        @print  W15msg
        @prx    4,<word ptr wbf.rcvbc+2>
        @prx    4,<word ptr wbf.rcvbc>
        @print  crlf

        @print  W16msg
        @prx    4,<word ptr wbf.rcve+2>
        @prx    4,<word ptr wbf.rcve>
        @print  crlf

        @print  W17msg
        @prx    4,<word ptr wbf.rtc+2>
        @prx    4,<word ptr wbf.rtc>
        @print  crlf

        @print  W18msg
        @prx    2,<word ptr wbf.xfmd-1>
        @print  crlf

        @print  W19msg
        @prx    2,<word ptr wbf.wtmd-1>
        @print  crlf

        @print  W20msg
        @prx    4,<word ptr wbf.extp>
        @print  crlf

        @print  W21msg                                          /Ma
        @prx    4,<word ptr wbf.xmtcol>                         /Ma
        @print  crlf                                            /Ma

        ret
hodat   endp

;-----------------------------------------------------------------------
savvecs proc    near
        push    ds
        push    es
        push    si
        push    di
        push    cx

        mov     ax,ds
        mov     es,ax
        xor     ax,ax
        mov     ds,ax
        mov     cx,22h*2        ;vectors 0 - 21h, 2 wds per
        mov     di,offset CODE:vectsv
        xor     si,si
        cld
        cli
   rep  movsw                   ;save 'em all
        sti

        pop     cx
        pop     di
        pop     si
```

```
        pop     es
        pop     ds
        ret
savvecs endp

;-------------------------------------------------------------------------
ixvecs proc     near
        push    es
        push    si
        push    di
        push    cx

        xor     ax,ax
        mov     es,ax
        mov     cx,22h*2        ;vectors 0 - 21h, 2 wds per
        mov     si,offset CODE:vectsv
        xor     di,di
        cld
        cli
  rep   movsw                   ;restore 'em all
        sti

        pop     cx
        pop     di
        pop     si
        pop     es
        ret
fixvecs endp

;-------------------------------------------------------------------------
;   dmprt - produces dump listing, calling parameters are pushed on stack
;           (converted from a C routine)
;   INPUTS:
;       [bp+4]  = data address
;       [bp+6]  = starting address for line headers
;       [bp+8]  = length of data to print
;   OUTPUT:
;       Dump listing to stdout device
;-------------------------------------------------------------------------
dmprt   proc    near

        push    bp
        mov     bp,sp
        mov     bx,bp
        sub     bx,0ch          ;local vars
        mov     sp,bx
        push    si
        mov     ax,[bp+8]       ;len

@0005c: sub     dx,dx
        mov     cx,10h

@0061:  div     cx
        mov     [bp-4],ax       ;lines

@0063:  mov     [bp-6],dx       ;rem

@0066:  mov     word ptr [bp-8],0       ;i
```

```
d0064:  mov     word ptr [bp-0ah],0     ;line

d0070:  jmp     d0158

d0073:
        push    dx
        mov     dl,cr           ;000d
        mov     ah,2
        int     21h
        mov     dl,lf           ;000A
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

        mov     ax,4
        push    ax
        mov     ax,[bp+6]       ;adr
        add     ax,[bp-8]       ;i
        push    ax
        call    prx
        add     sp,4            ;0004
        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

        mov     word ptr [bp-0ch],0     ;j

d00c5:  test    byte ptr [bp-0ch],3     ;j
        jnz     d00d5
        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

d00d5:  mov     ax,2            ;0002
        push    ax
        mov     bx,[bp-8]       ;i
        mov     si,[bp+4]       ;buf
        mov     ah,[bx+si]      ;buf[i]
        push    ax
        call    prx
        add     sp,4            ;0004
        inc     word ptr [bp-8] ;i
        inc     word ptr [bp-0ch]    ;j
```

```
d0010:  cmp     word ptr [bp-8],10h     ;
        jb      d00c5

        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

        sub     word ptr [bp-8],10h     ;i,0010
        mov     word ptr [bp-0ch],0     ;j


        ;do ascii
d0113:  mov     bx,[bp-8]               ;i
        mov     si,[bp+4]               ;buf
        push    dx
        mov     dl,[bx+si]              ;buf[i]
        cmp     dl,' '
        jb      d013f
        cmp     dl,7fh
        jb      d0142

d013f:  mov     dl,'.'                  ;002e

d0142:
        mov     ah,2
        int     21h
        pop     dx

        inc     word ptr [bp-8] ;i
        inc     word ptr [bp-0ch]       ;j
        cmp     word ptr [bp-0ch],10h   ;0010
        jb      d0113
        inc     word ptr [bp-0ah]       ;line

d0158:  mov     ax,[bp-4]               ;lines
        cmp     [bp-0ah],ax             ;line
        jnb     d0163
        jmp     d0073


d0163:  cmp     word ptr [bp-6],0       ;rem
        jnz     d016c
        jmp     d0272


d016c:
        push    dx
        mov     dl,cr                   ;000d
        mov     ah,2
        int     21h
        mov     dl,lf                   ;000a
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
```

```
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

        mov     ax,4            ;0008
        push    ax
        mov     ax,[bp+6]       ;adr
        add     ax,[bp-8]       ;i
        push    ax
        call    prx
        add     sp,4            ;0004
        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

        mov     word ptr [bp-0ch],0     ;j

        jmp     short   d01c3


d0198:  test    byte ptr [bp-0ch],3     ;j
        jnz     d01a8
        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

d01a8:  mov     ax,2            ;0002
        push    ax
        mov     bx,[bp-8]       ;i
        mov     si,[bp+4]       ;buf
        mov     ah,[bx+si]      ;buf[i]
        push    ax
        call    prx
        add     sp,4            ;0004
        inc     word ptr [bp-8] ;i
        inc     word ptr [bp-0ch]    ;j

d01c3:  mov     ax,[bp-6]       ;rem
        cmp     [bp-0ch],ax      ;j
        jb      d0198
        jmp     short   d01f4


d01cd:  test    byte ptr [bp-0ch],3  ;j
        jnz     d01dd
        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx
```

```
:01dd:
        push    dx
        mov     dl,'.'
        mov     ah,2
        int     21h
        mov     dl,'.'
        mov     ah,2
        int     21h
        pop     dx

        inc     word ptr [bp-0ch]    ;j

d01f4:  cmp     word ptr [bp-0ch],10h    ;0010
        jb      d01cd
        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx

        mov     ax,[bp-6]        ;rem
        sub     [bp-8],ax        ;i
        mov     word ptr [bp-0ch],0  ;j


        ;do ascii
d0219:  mov     ax,[bp-6]        ;rem
        cmp     [bp-0ch],ax      ;j
        jnb     d026c
        mov     bx,[bp-8]        ;i
        mov     si,[bp+4]        ;buf
        push    dx
        mov     dl,[bx+si]       ;buf[i]
        cmp     dl,' '
        jb      d024d
        cmp     dl,7fh
        jb      d0250

d024d:  mov     dl,'.'           ;002e

d0250:
        mov     ah,2
        int     21h
        pop     dx

        inc     word ptr [bp-8] ;i
        inc     word ptr [bp-0ch]    ;j
        jmp     short   d0219


d025f:
        push    dx
        mov     dl,'.'
        mov     ah,2
        int     21h
        pop     dx
```

```
        inc     [illegible]
d026c:  cmp     word ptr [bp-0eh],10h    ;001a
        jb      d025:

d0272:
        push    dx
        mov     dl,cr              ;000d
        mov     ah,2
        int     21h
        mov     dl,lf              ;000a
        mov     ah,2
        int     21h
        pop     dx

        pop     si
        mov     sp,bp
        pop     bp
        ret
dmprt   endp

;-----------------------------------------------------------------
;   prx - routine to print a hex value from binary data up to word length
;   INPUTS:
;       [bp+4]  = binary data to convert
;       [bp+6]  = number of bytes to print (1 to 4)
;
;-----------------------------------------------------------------
prx     proc    near

        push    bp
        mov     bp,sp
        mov     bx,bp
        sub     bx,4               ;local space
        mov     sp,bx

        push    si
        push    dx
        push    cx
        push    ds
        mov     ax,ss              ;make temp buf accessable
        mov     ds,ax
        lea     bx,[bp-4]          ;temp buffer address
        mov     dx,[bp+4]          ;data to cvrt
        call    wtoa
        mov     cx,[bp+6]          ;char count to print
        xor     si,si
prx1:
        mov     dl,[bp+si-4]       ;get a byte
        mov     ah,2
        int     21h                ;print it
        inc     si
        loop    prx1

        pop     ds
        pop     cx
        pop     dx
        pop     si
        mov     sp,bp
        pop     bp
```

```
;-------------------------------------------------------------
;       CONVERT WORD TO ASCII HEX
;    Calling sequence:
;       mov     dx,word         ;word to convert
;       mov     bx,offset out   ;where to put output
;       call    wtoa
;
;       ds:bx  needs 4 bytes for result
;-------------------------------------------------------------

wtoa    proc    near
        push    ax
        push    bx
        push    cx
        push    dx
        push    si
        mov     si,4            ;digits per word
wtoa01:
        mov     al,dl           ;get a digit
        mov     cl,4
        shr     dx,cl           ;strip the digit
        and     al,0fh          ;keep low nibble
        add     al,090h
        daa
        adc     al,040h
        daa
        dec     si              ;count the digit
        mov     [bx+si],al      ;store the digit
        jnz     wtoa01
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        ret
wtoa    endp

RCODE   ENDS
        END tstrx7              ;Ma
```